Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Удосконалення методу виявлення порушення конфіденційності файлів вихідного коду за рахунок збільшення інтервалів обробки та застосування транспозиції

Виконав: ст. 2-го курсу, групи УБ-21м
спеціальності 125– Кібербезпека
Освітня програма – Управління інформаційною безпекою
(шифр і назва напряму підготовки, спеціальності)

Ярова М. С.
(прізвище та ініціали)

Керівник: к.т.н., доц., зав. каф. МБІС
Карпінець В. В.
(прізвище та ініціали)
«15» грудня 2022 р.

Опонент: к.т.н., доц., доцент каф. ОТ
Крупельницький Л. В.
(прізвище та ініціали)
«15» грудня 2022 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

Юрій ЯРЕМЧУК
“15” грудня 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Управління інформаційною безпекою

ЗАТВЕРДЖУЮ
Голова секції УБ/ кафедра МБІС
_____**Юрій ЯРЕМЧУК**
“__15__” __вересня___2022 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Яровій Марії Сергіївні
(прізвище, ім`я, по-батькові)

1. Тема роботи Удосконалення методу виявлення порушення конфіденційності файлів вихідного коду за рахунок збільшення інтервалів обробки та застосування транспозиції
Керівник роботи к.т.н., доц., зав. каф. МБІС Карпінець В. В.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “14” вересня 2022 року № 203
2. Строк подання студентом роботи 12 грудня 2022 р.
3. Вихідні дані до роботи: Стандарти, монографії, підручники та наукові статті по темі. Існуюче програмне забезпечення, яке стосується теми магістерської дипломної роботи.
4. Зміст текстової частини: Для досягнення мети роботи було поставлено наступні задачі: зробити дослідження предметної області та дослідити існуючі проблеми доведення факту несанкціонованого копіювання файлів вихідного коду із застосуванням рефакторингу, дослідити існуючі алгоритми порівняння версій, розробити удосконалений метод для перевірки автентичності файлів вихідного коду на основі алгоритму порівняння версій, виконати програмну реалізацію запропонованого методу та провести його тестування.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
У першому розділі наведено 2 рисунки та 2 таблиці.
У другому розділі наведено 5 рисунків та 4 таблиці.
У третьому розділі наведено 4 рисунки та 3 таблиці.
У четвертому розділі наведено 6 таблиць.

У додатку Б наведено блок-схему алгоритму роботи розробленої підсистеми реалізації запропонованого рішення (1 рисунок).
У додатку Г наведено ілюстративний матеріал (презентація).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	Карпінець В. В., к.т.н., доц., зав. каф. МБІС		
Економічна частина	Лесько О. Й., , к.т.н., зав. кафедри ЕПВМ		

7.Дата видачі завдання 15 вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Початок роботи над МКР	15.09.2022	30.09.2022	
2.	Аналіз предметної області обраної теми	01.10.2022	10.10.2022	
3.	Апробація отриманих результатів	11.10.2022	15.10.2022	
4.	Розробка алгоритму роботи	16.10.2022	31.10.2022	
5.	Написання магістерської роботи на основі розробленої теми	01.11.2022	15.11.2022	
6.	Розробка економічної частини	15.11.2022	23.11.2022	
7.	Передзахист магістерської кваліфікаційної роботи	24.11.2022	25.11.2022	
8.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	26.11.2022	18.12.2022	
9.	Захист магістерської кваліфікаційної роботи	19.12.2022	21.12.2022	

Студент _____ (підпис) Ярова М. С.

Керівник роботи _____ (підпис) Карпінець В. В.

АНОТАЦІЯ

УДК 004.63

Ярова М. С. Удосконалення методу виявлення порушення конфіденційності файлів вихідного коду за рахунок збільшення інтервалів обробки та застосування транспозиції. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека, освітня програма – управління інформаційною безпекою. Вінниця, 2022. Кількість сторінок с.

На укр. мові. Бібліогр.: рис.: 12; табл. 15.

У магістерській кваліфікаційній роботі розроблено рішення для удосконалення методу порівняння файлів вихідного коду програм для виявлення факту несанкціонованого копіювання із використанням рефакторингу. У першому розділі проведено дослідження предметної області та проаналізовано існуючі методи для порівняння версій файлів вихідного коду. У другому розділі виконано розробку удосконалення для методу порівняння файлів вихідного коду програм на основі алгоритму для обчислення відстані Дамероу-Левенштейна із покращення показників точності порівняння версій та швидкості обробки алгоритмом вхідних даних. У третьому розділі розроблено програмну реалізацію удосконаленого методу та проведено тестування на предмет з’ясування досягнення планованого рівня показників точності та швидкості.

Графічна частина складається з 12 рисунків та 15 таблиць.

В економічній частині оцінено комерційний потенціал розробки та виконано розрахунки із прогнозування витрат на її реалізацію. Також, було виконано прогнозування комерційних ефектів від реалізації результатів розробки та розраховано ефективність вкладених інвестицій та період їх окупності.

Ключові слова: відстань Левенштейна, відстань Дамероу-Левенштейна, порівняння версій, рефакторинг, несанкціоноване копіювання, авторське право.

ABSTRACT

Yarova M. S. Improvement of the method of detection of violation of the confidentiality of source code files due to the increase of processing intervals and the use of transposition. Master's thesis on specialty 125 - Cybersecurity. Vinnytsia: VNTU, 2022. – Number of pages p.

In Ukrainian language. Bibliographer: fig.: 12; tabl. 15.

In the master's thesis, a solution was developed for improving the method of comparing the source code files of programs to detect the fact of unauthorized copying using refactoring. In the first chapter, the research of the subject area was carried out and the existing methods for comparing versions of source code files were analyzed. In the second section, an improvement was developed for the method of comparing source code files of programs based on the algorithm for calculating the Damerou-Levenshtein distance to improve the accuracy of version comparison and the speed of processing input data by the algorithm. In the third section, the software implementation of the improved method was developed and testing was conducted to find out whether the planned level of accuracy and speed indicators had been achieved.

The graphic part consists of 12 figures and 15 tables.

In the economic part, the commercial potential of the development was assessed and calculations were made to forecast costs for its implementation. Also, the forecasting of commercial effects from the implementation of the development results was performed and the effectiveness of the investments and their payback period were calculated.

Keywords: Lowenstein distance, Damerou-Levenstein distance, version comparison, refactoring, unauthorized copying, copyright.

ЗМІСТ

ВСТУП

Актуальність теми. Захист авторських прав на програмний код як об’єкт інтелектуальної власності у сфері розробки програмного забезпечення нині є серйозною проблемою для розробників, адже існуючі механізми захисту прав на володіння об’єктом інтелектуальної власності не є достатньо адаптованими до особливостей даної предметної області. Як не існує достатньо відповідних цим особливостям юридичних механізмів, так і програмних засобів, що дозволили б чітко визначати факт порушення автентичності файлів вихідного коду програм.

Оскільки програмний код з точки зору законодавства у сфері захисту авторського права розглядається як звичайний текст, то відповідно для пошуку ознак порушення автентичності файлів та відповідно доведення факту несанкціонованого копіювання із подальшим присвоєнням авторства використовуються ті ж рішення, що й для порівняння звичайних текстів. Однак особливості коду як текстової послідовності, (зокрема те, що зміни у ньому не так легко вносити як у звичайний текст, адже це може порушити працездатність коду) вимагають пошуку інших підходів, що враховуватимуть ці особливості.

Ще одним аспектом що є надзвичайно важливим для алгоритмів порівняння файлів вихідного коду є швидкість проведення перевірки. При проведенні судових чи внутрішніх службових розслідувань нині керуються набором методів та механізмів, що повинні виявляти характерний стиль автора і на основі цих доказів свідчити про автентичність чи не автентичність файлу. Однак, дані методи орієнтуються на ручне їх виконання, тож час обробки результатів таких досліджень є непристойно великий. Таким чином, існує гостра потреба у пошуку рішення, що дозволило б проводити перевірку автентичності файлів вихідного коду програм за значно менший час аніж існуючі методи та базуватиме свої висновки на певній чіткій метриці, що не даватиме простору для ентропії.

Метою дипломної роботи є удосконалення методу виявлення порушення конфіденційності файлів вихідного коду програм.

Завданнями дипломної роботи є:

1. Дослідити проблеми виявлення порушення автентичності файлів вихідного коду та дотримання авторського права у сфері розробки програмного забезпечення;

2. Дослідити можливості існуючих алгоритмів порівняння версій.

3. Удосконалити алгоритм порівняння версій з урахуванням особливостей предметної області.

4. Проаналізувати ефективність застосування розробленого рішення.

5. Розробити програмну реалізацію удосконаленого методу.

6. Протестувати удосконалений метод за допомогою розробленої його програмної реалізації.

Об’єкт дослідження – процес захисту від несанкціонованого копіювання порушення автентичності файлів вихідного коду програм.

Предмет дослідження – методи та засоби порівняння версій файлів вихідного коду програм.

Новизна одержаних результатів. Застосовано метод порівняння версій файлів на основі алгоритму для визначення відстані Дамероу-Левенштейна у нетиповій предметній області та розроблено удосконалення до його реалізації для даної предметної області.

Практичне значення одержаних результатів. Розроблено програмну підсистему для порівняння версій файлів з метою доведення чи спростування факту порушення автентичності файлів вихідного коду програм та вчинення факту несанкціонованого копіювання, що також може бути інтегрована до більшої програмної системи захисту.

Публікації. За результатами дослідження було опубліковано 2 тез доповідей.

Апробація. Результати дослідження було оприлюднено на L науково-технічній конференції підрозділів Вінницького національного технічного університету (м. Вінниця, 2021 р.). та Всеукраїнській науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» (м. Вінниця, 2022).

1 ДОСЛІДЖЕННЯ ПРОБЛЕМ ВИЯВЛЕННЯ ПОРУШЕННЯ АВТЕНТИЧНОСТІ ФАЙЛІВ ВИХІДНОГО КОДУ ТА ДОТРИМАННЯ АВТОРСЬКОГО ПРАВА У СФЕРІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі буде розглянуто наявні проблеми захисту авторського права на інтелектуальну власність у сфері розробки програмного забезпечення, а також актуальних нині засобів та методів, що застосовуються для проведення розслідування фактів несанкціонованого копіювання файлів вихідного коду програм. Буде проаналізовано умови, що роблять ці методи неефективними у розрізі предметної області що розглядається, а також досліджено критерії яким має відповідати алгоритм, щоб його можна було назвати правильним рішенням для обраної для даної роботи задачі.

1.1 Проблеми існуючого стану захисту авторського права у сфері розробки програмного забезпечення

Першочерговою проблемою авторського права є те, що воно розроблялось без урахування сфери розробки програмного забезпечення та її специфіки. Цим можна пояснити зокрема неоднозначність визначення комп’ютерної програми у правовому полі.

Більшість світових законодавств у галузі захисту авторського прав базуються на домовленостях, що описані у Бернській конвенції [1]. Згідно даної конвенції, комп’ютерна програма розглядається як літературний твір і до неї, відповідно, застосовуються ті ж норми захисту авторського права, що й до текстових творів. Україна є членом Бернської конвенції, що знайшло своє відображення у статті 8 та 18 Закону України «Про авторське право та суміжні права» [2]: «Комп’ютерні програми охороняються як літературні твори. Така охорона поширюється на комп’ютерні програми незалежно від способу чи форми їх вираження». Згідно Закону України «Про авторське право і суміжні права», «комп’ютерна програма - набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи у будь-якому іншому вигляді, виражених у формі, придатній для зчитування комп’ютером, які приводять його у дію для

досягнення певної мети або результату (це поняття охоплює як операційну систему, так і прикладну програму, виражені у вихідному або об’єктному кодах)».

Потенційно таке визначення можна застосувати до звичайних програм, що призначені лише для запуску на локальній станції. Однак трактувати такі норми законодавства для об’ємних ІТ-проектів, мобільних додатків, масштабних ігор тощо уже не так просто. Якщо розглядати середньостатистичний комерційний проект звичайної ІТ-компанії можна виділити як мінімум 6 об’єктів, що можна характеризувати як окремі одиниці захисту авторського права: вихідний код, бази даних, об’єктний код, алгоритми, інтерфейс та програмно апаратні комплекси [3]. Як бачимо, далеко не усі з них можна охарактеризувати як літературний твір.

Однак, слід віддати належне, що правовою системою все ж передбачено альтернативне рішення, що може за певних умов захистити права розробника. Різні компоненти програмного забезпечення, окрім авторського права, можуть бути захищені патентами.

Принципова різниця між авторським правом та патентом полягає у об’єкті захисту та моменті ініціалізації прав власника у правовому полі. Авторське право захищає лише зовнішнє представлення об’єкта інтелектуальної власності. Це підкреслено у вище згадуваній ст. 8 Закону України «Про авторські та суміжні права»: «Передбачена цим Законом правова охорона поширюється тільки на форму вираження твору і не поширюється на будь-які ідеї, теорії, принципи, методи, процедури, процеси, системи, способи, концепції, відкриття, навіть якщо вони виражені, описані, пояснені, проілюстровані у творі». Натомість, патент охороняє суть розробки, конкретне рішення, що лежить в її основі. Авторське право виникає автоматично з моменту створення об’єкта інтелектуальної власності і є актуальним на території усіх країн-членів вище згадуваної Бернської угоди, натомість патентування здійснюється в результаті державної реєстрації. Однак, варто враховувати, що оформлення патенту потребує від об’єкту патентування відповідності конкретним вимовам, згідно яких його можна назвати унікальним з точки зору технічної розробки [4].

Якщо повернутись до компонентів ІТ-проекту, що потребують захисту як об’єкти інтелектуальної власності, то авторським правом охороняються вихідний код (як літературний твір) та об’єктний код (як перетворений вихідний код) і бази даних (окремо визначені об’єктом авторського права). Алгоритм же є не просто набором записів, він описує чіткий порядок логічних операцій, які повинен виконувати комп’ютер при роботі програми. Саме тому, алгоритм як певну послідовність операцій може бути визначено як винахід і запатентовано. Якщо ж говорити про інтерфейс, то його можна запатентувати як алгоритм, якщо розглядати в контексті процесу взаємодії користувача з програмою, або як промисловий зразок, якщо говорити про його зовнішню складову. Програмно-апаратний комплекс може піддаватись захисту через патентування як винахід або модель.

Попри деякі очевидні переваги патентування у сфері захисту прав на інтелектуальну власність, у цього методу є істотний недолік: територіальна сфера дії патенту обмежується країною, у якій його було зареєстровано. Тобто на території іншої країни патент може не мати ніякої юридичної сили і довести авторство скопійованої розробки за межами країни у якій її було запатентовано буде неможливо. А якщо врахувати те, що більшість сучасного програмного забезпечення розробляється розподіленими командами, у яких розробники працюють з різних куточків не тільки країни, а навіть світу, то патентування є незручним, громіздким та занадто вже ненадійним методом захисту авторського права в контексті компонентів програмного забезпечення.

Інша сторона проблем захисту авторського права у сфері розробки програмного забезпечення пов’язана з визначенням безпосередньо автора твору, тобто кому належать авторські права. На сьогоднішній день розробка одного програмного продукту передбачає роботу над його реалізацією та випуском у масове користування певної кількості зацікавлених сторін. Власне кажучи, усі хто бере участь у розробці програмного продукту можуть претендувати на авторські права щодо нього: розробники-програмісти, підрядники, клієнти-замовники та навіть компанія в цілому, як юридична особа. Те, на чиїй стороні опиниться судова система при вирішенні суперечливих питань авторського права, залежить від того, наскільки

послідовно підійшли розробники до захисту своїх інтересів при підписанні договорів та ліцензій на використання програмного забезпечення. Згідно авторського права, права автора твору (в даному випадку деякої програмної розробки чи коду) поділяються на майнові та немайнові.

Немайнові авторські права є невід'ємними від автора розробки. Хоч обсяг цих прав може відрізнятись залежно від країни, у якій власник інтелектуальної власності намагається захистити свої інтереси, в цілому, вони зберігаються за автором, він не може передати їх іншим особам чи бути позбавлений їх примусово. Немайнові права включають в себе право бути вказаним як автор чи право на заборону зміни твору (в даному випадку програмного продукту чи його частин), якщо це спотворює його зміст, функції тощо. Для прикладу, у випадку, коли програмний продукт було розроблено на замовлення, розробник, що є автором проекту, залишає за собою певний обсяг немайнових авторських прав на цю інтелектуальну власність, незалежно від того, чи прописані у договорі із замовником умови передачі авторських прав на розробку.

Натомість, майнові права надають автору виключне право на використання розробки та право забороняти його використання сторонніми особами. Лише автор може надавати іншим особам дозвіл на використання продукту чи його частин, зокрема, поширювати, об'єднувати з іншими програмними продуктами, модифікувати, адаптувати тощо. Майнові права можна передавати лише вдаючись до наступних прийомів: або через оформлення договору про надання послуг (якщо говорити про передачу авторських прав від розробника клієнту-замовнику) чи трудового контракту (в контексті визначення прав інтелектуальної власності між розробником програмного продукту та компанією, що його найняла), або за допомогою ліцензії (відкритої чи комерційної). І саме на цьому моменті виникають основі суперечності з приводу визначення авторських прав на програмний продукт [5, 6].

Увесь обсяг коду програмного продукту, якщо розглядати його з точки зору можливостей та процедур передачі авторських прав, умовно можна поділити на три типи: існуючий (наявний) код, відкритий код та унікальний код (рисунок 1.1).

Рисунок 1.1 – Поділ вихідного коду проекту програмного продукту з точки зору присвоєння авторського права

Відкритий код (Open-source Code) знаходиться у загальному доступі та може піддаватись модифікації і вільно розповсюджуватись. Ні розробник, ні клієнт-замовник не можуть претендувати на авторські права на цей код. Зазвичай автори відкритого коду вдаються до захисту своїх немайнових прав і поширення такого програмного забезпечення здійснюється на умовах відкритих ліцензій. Попри те, що відкриті ліцензії фактично захищають лише немайнові права автора, не знання чи порушення умов відкритої ліцензії може нести вагомі фінансові ризики для комерційного проекту. Наприклад, в умовах деяких з них вказано необхідність опублікування вихідного коду повного проекту, у якому було використано захищений даною ліцензією відкритий код. Тому, програміст, що хоче використати код з відкритою ліцензією для власної розробки повинен запевнитись, що має право використовувати його і що усі вимоги ліцензії виконуються належним чином. Клієнт-замовник обов’язково повинен знати який код у проекті є відкритим та які обмеження ліцензії щодо його використання існують. З точки зору розробника відкритого коду, також існують певні недоліки захисту за допомогою відкритих ліцензій. Для

прикладу, залежно від особливостей законодавства, умови відкритих ліцензій можуть втрачати свою юридичну силу, якщо не підпадатимуть під звід правил щодо форм та змісту ліцензії. Так, згідно Закону України «Про авторське право та суміжні права», обов'язковою вимогою для чинності ліцензії є її представлення у письмовій формі. Тобто, звична форма укладання договору згоди на виконання ліцензійних умов у електронному вигляді з юридичної точки зору не можу слугувати доказовою базою.

Існуючий код (Existing/Third-party Code) – це код, що був написаний розробниками компанії для інших проектів раніше. Компанія, що виступає постачальником послуг з розробки програмного забезпечення зазвичай не зацікавлена у відмові від прав власності на даний код, оскільки може використовувати його для реалізації певних функцій при створенні інших продуктів. Тому, для захисту своїх прав на існуючий код, компанії зазвичай пропонують своїм замовникам використання програмного продукту на умовах комерційної ліцензії.

Унікальний код (Unique Code) – це той код, який був розроблений для реалізації конкретного проекту. Зазвичай, компанія замовник хоче отримати усі права власності на придбану розробку, тож ці умови прописуються або у договорі на замовлення продукту, або затверджуються шляхом видачі на замовника ексклюзивної ліцензії. У будь-якому випадку, в результаті клієнт стає фактичним власником авторських прав на унікальний код, а отже може комерціалізувати його у своїх цілях [7].

Описані вище засоби захисту при системному та послідовному підході дозволяють компаніям-постачальникам послуг та їх клієнтам надійно захистити інтереси кожної зі сторін, у випадку вирішення суперечок навколо авторських прав на програмний продукт.

Найскладнішим питанням для розробників є захист від несанкціонованого копіювання безпосередньо коду. Під несанкціонованим копіюванням розуміють відтворення об'єкту копіювання з перевищенням встановлених власником прав доступу на використання. З точки зору програмного забезпечення, говорять про несанкціоноване копіювання вихідного коду, об'єктного коду та самої розробки. Копіювання об'єктного коду більше пов'язане із поширенням піратського програмного забезпечення і у більшості світових законодавств уже передбачені

норми захисту від нього. З точки зору несанкціонованого копіювання розробки (тобто ідеї, алгоритму), як уже було зазначено раніше, якщо розробник хоче захистити свою ідею та однозначно закріпити за собою права на проект, йому слід потурбуватись про оформлення патенту на розробку.

Існує безліч ситуацій, коли необхідно просто однозначно ідентифікувати авторство коду, у тому вигляді, у якому його було написано.

Та спершу, для розуміння механізмів захисту прав автора при несанкціонованому копіюванні написаного ним програмного продукту, потрібно з'ясувати, що з точки зору юриспруденції вважається копіюванням вихідного коду. Копіювання коду може виражатись у двох формах:

— пряме запозичення – копіювання програмного коду відбувається тією ж мовою програмування, з внесенням незначних змін, додаткового функціоналу, з метою приховування оригінальних авторських прав на програмне забезпечення;

— непряме запозичення – копіювання здійснюється іншою мовою програмування і запозичуються лише окремі частини розробки: структура, деякі функції, операційні послідовності тощо.

З точки зору непрямого запозичення, оцінити шанси позивача досить не складно – тут найважливіше те, які аргументи він представляє. За таких умов краще не посилатись на виконання копією програми таких же функцій, що й у оригінального продукту. Адже авторське право не захищає унікальність розробки на рівні ідеї, а отже випуск нової програми з подібним функціоналом не може вважатись порушенням авторського права, якщо лише на розробку не було оформлено патент.

Вирішення суперечок при оскарженні факту прямого запозичення відрізняється більшою невизначеністю. У судовій практиці розв'язання справ незаконного прямого копіювання вихідного коду вдаються до застосування критеріїв за допомогою яких порівнюють версії коду, що проходить у справі: абстрагування, фільтрація та порівняння. Абстрагування передбачає перегляд обох версій вихідного коду (оригінального і запозиченого) з найнижчого рівня до найвищого: порівняння їх від коду безпосередньо до суті функцій, що вони виконують. При фільтрації з продукту забирають усі ті частини, що не підлягають захисту з точки зору авторського права

(зазвичай, частини відкритого коду), стандартні для даної мови програмування та типу задачі процедури (наприклад, оголошення змінних на початку) та ті, що захищаються патентним правом або визначені комерційною таємницею. Відкинувши всі вище вказані частини, до розгляду суду представляються елементи, що можуть розглядатись в рамках закону про авторське право. Порівняння – це є процес безпосереднього зіставлення версій вихідного коду. На цьому етапі визначають наскільки велику частину становить вміст оригінального коду у копії.

Перевірку версій коду виконують на вимогу суду експерти-спеціалісти, які можуть для більшої об’єктивності аналізу вимагати інші зразки коду автора для порівняння наприклад характеру, стилю написання коду. Та за сучасних умов, коли роботу над проектом зазвичай веде декілька розробників, ефективність та прозорість таких перевірок значно знижується [8]. Також варто враховувати і те, що протягом розробки код може проходити поверхневу структурну обробку з метою уніфікації, спрощення чи підвищення ефективності коду, що також повністю стирає будь-який натяк на характерний стиль автора коду.

Таким чином, можна зробити висновок що якогось одного шаблону для оформлення захисту прав на інтелектуальну власність розробнику існуючою міжнародною системою захисту авторських прав не передбачено. Єдине на що може опиратись розробник за таких умов – це правильно складені умови контракту із компанією та із замовником, в контексті розподілу та порядку передачі прав на володіння об’єктом інтелектуальної власності у вигляді вихідного коду програми. Однак не вирішеним залишається питання того, як однозначно закріпити авторство версії коду за конкретним розробником та забезпечити доказову базу для випадків несанкціонованого копіювання, адже як уже було з’ясовано, методи, що нині практикуються при розслідуванні таких випадків є абсолютно неефективними та не здатні забезпечити достатню долю об’єктивності.

1.2 Вивчення принципів за якими будуються сучасні підсистеми виявлення порушення автентичності файлів вихідного коду

Розробка програмного забезпечення на сьогоднішній день практично не являється одноосібною задачею: в більшості своїй до цього процесу залучена велика команда розробників, які до того ж нерідко є розосередженими по різних містах чи навіть країнах. Будь-яка компанія чи навіть команда розробників-фрілансерів зацікавлені у тому щоб їхня робота зберігалась з абсолютним дотриманням цілісності доступності та конфіденційності. Зокрема, важливим аспектом є можливість збереження версій коду у часі, запис авторства, а також відслідковування автентичності файлів вихідного коду. Засобом, що поєднує у собі увесь названий вище функціонал стали системи контролю версій – програмний інструмент, що дозволяє реєструвати, контролювати та документувати зміни у файлах коду. Вони необхідні для злагодженої роботи над проектом групи розробників та забезпечення цілісності кодової бази. Усі розробники мають доступ до спільного репозиторію (сховища) проекту та можуть працювати в ньому одночасно, незалежно від їх місцезнаходження. Системи контролю версій зберігають історію змін та їх авторів і таким чином, дають можливість повертатись до більш ранніх версій файлів, полегшувати процес пошуку помилок, відслідковувати вклад кожного розробника при розробці програмного продукту, тож їх іноді розглядають і як інструмент резервного копіювання. Окремі файли проекту в таких системах розподіляються за ієрархічною деревоподібною структурою, завдяки чому, окремі потоки роботи можуть виконуватись паралельно. А спільний доступ до репозиторію дозволяє різним розробникам запобігати порушень відносно вихідного коду, написаного їх колегами [9, 10].

Перераховані вище функції є не єдиною причиною того, чому без системи контролю версій не обходиться робота у жодному проекті зі створення програмного забезпечення. Використання такого спільного інструменту розробки забезпечує аудит інтелектуальної власності у проекті, в тому числі, з точки зору порушення авторського права. Ця функція є надзвичайно важливою, адже поява коду, що порушує норми авторського права в одному місці програми, може вплинути на

законність використання більшості інших складових компонентів системи. Для прикладу, розробник може вносити у репозиторій код, який він запозичив з відкритого доступу, при цьому він може навіть не здогадуватись, що порушує правові норми захисту інтелектуальної власності, особливо якщо останнє було захищене патентним правом чи ліцензійними умовами. Також, виконання задач ідентифікації авторства є актуальними з точки зору потенційного виникнення суперечок всередині команди розробників з приводу авторства коду та правок до нього [11].

В цілому, усі інструменти контролю версій початково передбачають виконання функцій визначення авторства та порівняння версій файлів вихідного коду, різниця полягає скоріше у механізмах ідентифікації автора та їх надійності. Для того щоб зрозуміти які методи та алгоритми на сьогоднішній день використовуються для забезпечення безпеки конфіденційності файлів вихідного коду, та відповідно, мати можливість проаналізувати їх ефективність, розглянемо декілька найбільш затребуваних систем контролю версій.

Першою варто розглянути одну з ранніх, проте досі актуальних систем – Subversion (нині більш відома як Apache). Вона належить до централізованих систем контролю версій, що означає, що усі файли зберігаються та знаходяться під контролем версій на центральному сервері. Усі учасники проекту (розробники) отримують копії файлів з нього. Організація репозиторіїв може здійснюватися двома шляхами: зберігання звичайних файлів у спеціальному форматі, або зберігання бази даних файлів. Для вирішення конфліктів при внесенні змін до одного і того ж файлу, система дозволяє додавати їх лише в останню версію вихідного коду. В історії змін відображаються дані про абсолютно усі маніпуляції, що були виконані з файлом (наприклад, перейменування, копіювання, переміщення). Контроль доступу користувачів до файлів у репозиторії здійснюється за рахунок алгоритму на основі шляху до файлів (Path-Base Access Control). За замовчуванням, усі кому надано доступ до репозиторію отримують можливість редагування усіх файлів у ньому, незалежно від сфери їх діяльності. Адміністратор безпеки має можливість налаштувати права доступу кожного користувача до різних гілок та файлів на сервері: обмежити доступ, змінити права доступу у тих областях проекту, у перегляді та

редагуванні яких конкретний розробник не має потреби, у рамках виконання своїх службових обов’язків у проекті. Однак, для обробки налаштувань прав доступу користувача серверу потрібно затрачати значно більше ресурсів. У зв’язку з цим відмічається значна втрата у продуктивності роботи усієї системи. Тому, за таких умов, адміністратору потрібно оцінити необхідність налаштувань доступу для конкретного проекту (інакше кажучи, наскільки висока імовірність того, що якийсь з членів команди скористається своїм загальним доступом з поганими намірами). Цей факт можна відмітити як серйозний недолік, що обмежує гнучкість при забезпеченні безпеки коду.

Ще одним із серйозних недоліків Subversion та усіх централізованих систем в цілому є вразливість сервера, що є центральним місцем у таких системах. При технічних збоях чи пошкодженні диску та центральної бази даних і відсутності резервного копіювання втрачається абсолютно вся інформація з репозиторію (версії файлів, історія). Фактично, зловмисник може здійснити акт несанкціонованого копіювання, а потім зламати репозиторій. За таких умов, довести порушення авторських прав уже не вдасться, адже уся доказова база буде втрачена разом із базою даних версій та історією змін, де власне відображаються дані, що можуть засвідчити оригінальність та авторство вихідного коду [12, 13].

Описана вище проблема була вирішена із появою розподілених систем контролю версій. При кожному доступі до перегляду чи редагування файлів у репозиторії користувач не просто завантажує останні їх версії, а копіює усе сховище повністю. За таких умов, у разі пошкодження серверу, система може відновити репозиторій, скопіювавши дані з клієнтського диску. Зазвичай набір версій розподіляється між декількома сховищами на боці користувачів, що ідеально вписується у принципи віддаленої роботи над проектом та є дуже популярною в останні роки. Будь-який розробник може вивантажити собі останнє оновлення версій і продовжити роботу на локальній машині, а потім завантажити виконані ним оновлення назад у репозиторій. Це дуже зручна особливість, особливо якщо є необхідність виконати роботу за відсутності Інтернет з’єднання. Та з іншого боку, за таких умов, будь-хто із розробників може вчинити несанкціоноване копіювання

спільної роботи і видати проект за власну розробку. Наявність у оригінальних авторів доказової бази залежатиме від етапу розробки проекту, на якому було вчинено даний акт копіювання, доопрацювання змін у коді, які виконав зловмисник, а також частково, особливостей системи авторизації та контролю доступу конкретної розподіленої системи контролю версій.

Серед розподілених систем контролю версій варто розглянути Mercurial та Git. Системи були випущені приблизно в один період і обидві мають відкритий код.

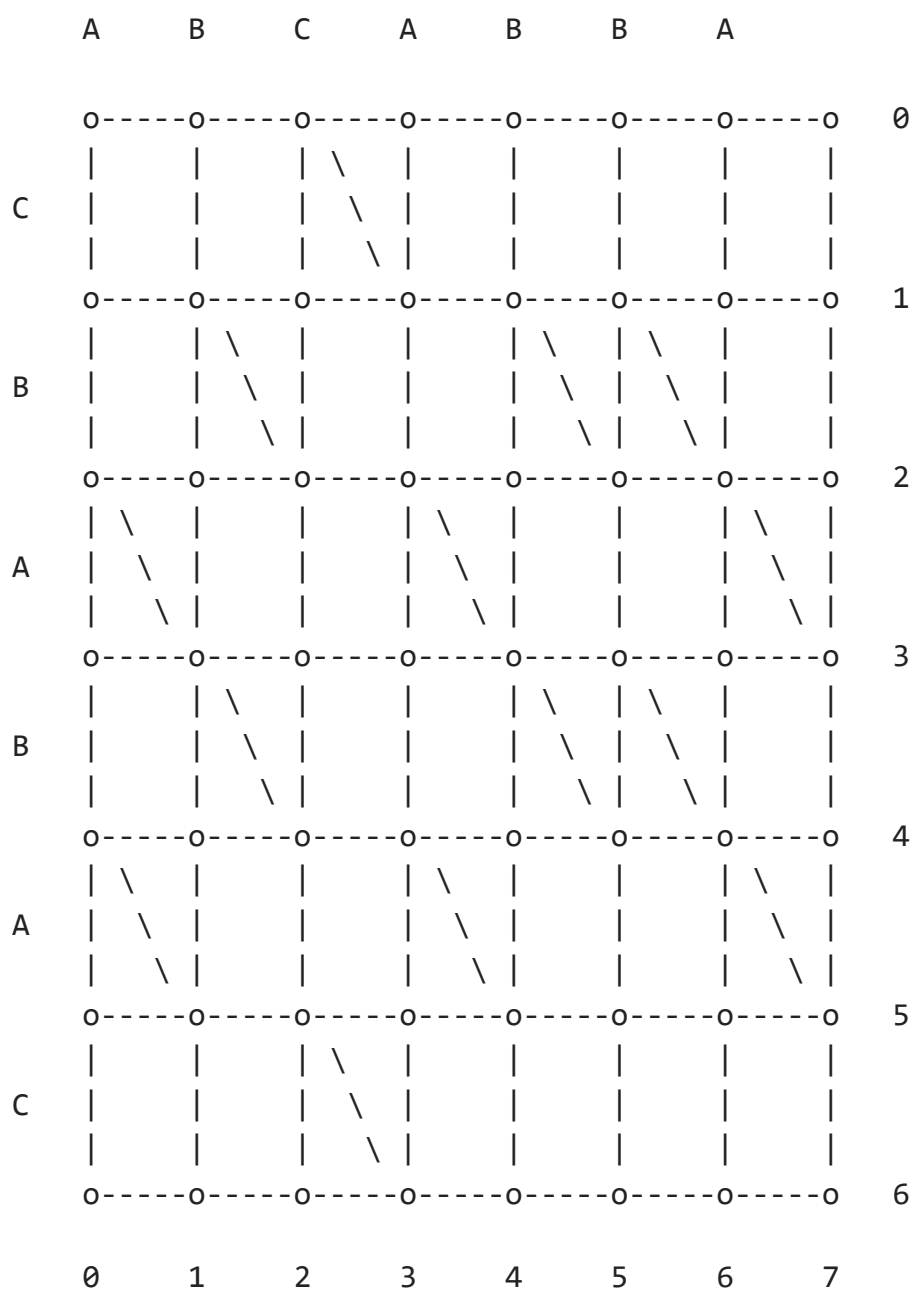
Mercurial відрізняється високою індексацією та компактним зберіганням даних. Продуктивність системи не залежить ні від зайнятого об'єму сховища, ні від числа зафіксованих змін, ні від налаштувань управління доступом. Організація репозиторію відбувається по типу журналу: дані про зміни не заміняють попередню версію, а додаються до неї. Також, додатково ведеться журнал переміщення файлів. Цілісність файлів захищається за рахунок застосування алгоритму SHA1 та вбудованих інструментів її перевірки. Mercurial також містить вбудовані інструменти для виконання резервного копіювання [14,15].

Git вважається нині найпродуктивнішою та найбільш надійною з систем контрою версій. Подібно до Mercurial, дані про внесені у файл зміни зберігаються у сховищі через систему «зліпків»: після кожної внесеної зміни у сховищі зберігається набір усіх змін файлу. Порівняння версій файлів виконується за алгоритмом Майерса, або можливий вибір у сторону розширеного алгоритму гістограми.

Даний алгоритм було запропоновано у 1986 Юджином Майєрсом. Він покращив популярну на той час утиліту diff, інструмент порівняння даних, який відображав найменший набір построкових видалень і вставок для перетворення одного файлу в інший.

Згідно алгоритму, щоб сформувати редагований графік слід розташувати вихідну послідовність зліва направо вздовж осі X, а послідовність призначення зверху вниз вздовж осі Y з ортогональною сіткою горизонтальних і вертикальних країв, спроектованих з них. Починаючи з верхнього лівого кута і шукаючи нижній правий кут, кожен рух уздовж країв графа до вершини відповідатиме інструкції редагування вихідної послідовності. Рух праворуч (збільшення X) відповідає видаленню символу

з a, наприклад, переміщення до (1,0) означає, що ми видалили першу A з a. Переміщення вниз (збільшення y) відповідає вставці символу з b, наприклад, якщо ми тепер рухаємося від (1,0) вниз до (1,1), ми вставляємо перший C з b, і таким чином наш відредагований рядок буде CBCABBA. У позиції (4,3) ми перетворили ABCA на CBA, але нам ще потрібно перетворити BBA на BAC. Нижня права позиція (7,6) відповідає повному перетворенню рядка a в рядок b.



Ідея алгоритму Майєрса досить проста: ми хочемо дістатися від (0,0) до (7,6) (внизу праворуч) за якомога менше ходів. «Хід» — це один крок праворуч (видалення з a) або вниз (вставка з b). Найбільша кількість ходів, яку ми можемо зробити, щоб

перейти від a до b, становить 13: сумарна довжина двох рядків. Однак ходіння по діагональних шляхах є безкоштовним, оскільки вони не відповідають внесенню змін, тому ми хочемо максимізувати кількість кроків по діагоналі, які ми робимо, і мінімізувати кількість рухів праворуч/вниз [16].

Цілісність даних в Git також забезпечується завдяки алгоритму SHA1. Для формування індексу версії використовуються дані її файлу. Отриманий SHA-1 хеш стає індексом версії файлу. Для підвищення безпеки автентифікації користувачів, передбачена можливість прив'язки змін, що вноситься розробниками до їх електронної пошти чи електронного підпису. Таким чином Git забезпечується гарантування автентичності змін вихідного коду. Це створює величезну перевагу Git серед інших розподілених систем, у яких засвідчення авторського права версій відбувається або лише всередині системи, або на дуже слабкому рівні [17, 18].

Все ж варто розуміти, що жодна з досі відомих систем контролю версій не дозволяє проводити розслідування автентичності коду: ті алгоритми що у них застосовані спрямовані виключно на забезпечення того щоб система могла відрізнити версії коду у часі (тобто для функціонування опції збереження історії). Для того щоб система дозволяла виявляти схожість та однозначно вказувати на оригінал необхідне поєднання декількох компонентів. Перш за все, використання цифрових водяних знаків, як засобу для однозначного присвоєння авторства коду. За допомогою цифрових водяних знаків теоретично можна розглянути реалізацію запису та обліку історії версії, адже окрім даних про автора до складу даних цифрового водяного знаку можна додавати час створення файлу. Другим складником, звісно ж має стати алгоритм для порівняння версій файлів. Під методом порівняння версій вихідного коду слід розуміти алгоритм, що дозволятиме виявляти подібність копії файлу та його оригіналу, навіть за умови застосування до копії прийомів рефакторингу. На жаль, сучасні системи контролю доступу станом на сьогодні обходяться примітивним методом пошуку по точному співпадінню – алгоритмом пошуку найдовшої співпадаючої послідовності (Longest Common Subsequence, LCS). Порівняння версій файлу за цим алгоритмом полягає у пошуку частин, які було змінено у копії, а які – ні.

Реалізація алгоритму потребує високих затрат пам’яті, тому в якості одиниць порівняння використовують абзаци. В рамках алгоритму виділяють три статуси стрічки коду: незмінна (є однаковою у обох версіях файлу, що порівнюються), видалена (наявна лише у старшій версії, що розглядається як оригінальна) та додана (присутня лише у новішій версії, що розглядається як можлива копія файлу). Стрічка вважається незмінною, якщо міститься і є однаковою в обох версіях файлу. Доданою називають ту стрічку, яка відсутня у старшій версії (тій, що розглядається як оригінальна), але присутня у новішій (тій, що розглядається як можлива копія). І навпаки, видалена стрічка та, яка є у першій версії, але відсутня у новій.

У різних системах реалізовано різні рівні складності обрахунків даного алгоритму: у деяких це складні логарифмічні обчислення, а у деяких складність обрахунку залежить напряму від довжини порівнюваних кодових послідовностей. У будь-якому випадку, порівняння версій файлів вихідного коду майже не виконується безпосередньо по-стрічково (хоча відомі випадки і такої реалізації). Зазвичай, йдуть більш зручним шляхом з використанням хеш-функцій: перед порівнянням зі стрічок отримують хеш і порівнюють самі хеші. У випадку співпадіння хешів, додатково повторюють перевірку уже на рівні стрічок. Таким чином, обходять похибку, яку можуть спричинити колізії при хешуванні.

Даний алгоритм є достатнім для виконання функцій системи управління доступом, однак є абсолютно неефективним для порівняння оригіналу файлу та його копії після застосування до останньої прийомів що змінюють зовнішній вигляд коду. Зміна зовнішнього вигляду коду може мати багаторівневий характер: від простої перестановки рядків місцями, до згортання модулів, перейменування змінних і класів тощо. Такою технікою, що використовується для заплутування та видозміни коду є рефакторинг.

В цілому, рефакторинг – це практика покращення програмного коду. Вона передбачає його перетворення, зміну на рівні внутрішньої структури. При цьому існуючий функціонал продукту залишається без змін. Початково, його метою є підвищення ефективності коду, за рахунок перетворення складного масивного коду у більш простий, без втрати продуктивності його роботи. До рефакторингу програмісти

вдаються на етапі розробки чи тестування, у випадках коли потрібно усунути проблему некоректної роботи коду, підвищити його здатність до масштабування, адаптивність чи навіть швидкість та ефективність роботи [20].

При застосуванні технології рефакторингу зовнішній вигляд в більшій чи меншій мірі змінюється, чим охоче користуються зловмисники. Ступінь зміни вигляду вихідного коду залежить від виду прийомів рефакторингу, що були застосовані та ітеративності їх використання. Тому необхідно дослідити наскільки змінюється зовнішній вигляд коду при використанні тих чи інших методів та відповідно наскільки ефективними вони є, з точки зору приховування факту запозичення.

У таблиці 1.1 наведено порівняльну характеристику методів рефакторингу з цієї точки зору [21].

Таблиця 1.1 – Порівняльна характеристика методів рефакторингу з точки зору зміни зовнішнього вигляду вихідного коду

Група методів	Техніки методу	Рівень зміни зовнішнього вигляду коду
1	2	3
Складання методів	Виокремлення та вбудовування методів, виокремлення та вбудовування змінних, заміна змінної викликом методу, розщеплення змінної, заміна методу об’єктом методів, заміна алгоритму роботи методу, видалення присвоєнь параметрам	Слабкий

Продовження таблиці 1.1

1	2	3
Переміщення функцій по класах	Переміщення методу і поля, відокремлення, вбудовування класу, приховання делегування, видалення посередника, введення зовнішнього методу та введення локального розширення	Середній
Організація даних	Заміна значення посиланням та посилання значенням, дублювання видимих даних, самоінкапсуляція поля, заміна простого поля об’єктом, заміна поля-масиву об’єктом, заміна підкласу полями, інкапсуляція поля та колекції тощо	Високий
Спрощення умовних виразів	Об’єднання та розподіл умовних операторів, заміна умовних операторів поліморфізмом, введення Null-об’єктів тощо	Слабкий

Продовження таблиці 1.1

1	2	3
Спрощення викликів взаємодії	Додавання та видалення параметрів, перейменування методів, розділення запитів і модифікаторів, параметризація методів, заміна параметрів об’єктами, наборами спеціалізованих методів, викликами методу тощо	Слабкий
Задачі узагальнення об’єктів	Підйом поля, методу, тіла конструктора, спуск поля, методу, відокремлення інтерфейсу, створення шаблонного методу, заміна наслідування делегуванням та делегування наслідуванням тощо	Середній

В цілому, виявити факт копіювання вихідного коду при застосуванні рефакторингу є можливим у разі, коли оригінальна версія була збережена у початковому вигляді (у якому її створив розробник), а її копія була створена із застосування прийомів рефакторингу. В такому випадку, при порівнянні версій коду буде помітним характерний стиль написання розробника, послідовність обробки коду, оскільки більшість методів рефакторингу з точки зору зовнішнього вигляду вносять лише поверхневі зміни. Якщо ж оригінальна версія була випущена уже із застосуванням рефакторингу (наприклад, його було застосовано під час тестування

чи на іншому етапі розробки), а зловмисник піддав її рефакторингу знову, тим самим чи іншим набором методів, грань показників авторства дещо втрачається. Тут на перший план виступає ітеративність процедури рефакторингу: з кожною зміною коду його подібність до оригіналу стає все меншою. Ускладнює ситуацію використання автоматизованих засобів виконання рефакторингу. Вони можуть бути і як у вигляді окремих додатків, так і бути вбудованим налаштуванням середовища розробки. За їх використання характерні ознаки стилю написання коду автором зникають ще на етапі розробки.

Таким чином, порівняння версій файлів можу бути ефективним методом виявлення порушення автентичності лише у випадку застосування до оригінального коду простих методів рефакторингу на кшталт перейменування змінних, перестановки рядків тощо. Однак, навіть за таких умов, алгоритм пошуку найдовшої співпадаючої послідовності є абсолютно програшним варіантом. Опиратись лише на визначення того чи є стрічка незмінною, доданою чи видаленою, буде абсолютно програшною тактикою, оскільки в даному випадку алгоритм порівняння має проводити глибокий аналіз подібності двох версій абзацу стрічок та оцінювати ступінь запозичення. Тож, існує закономірна причина для пошуку алгоритму, що дозволить обходити заплутування та зміну коду уже на рівні застосування найпростіших форм рефакторингу.

1.3 Дослідження алгоритмів порівняння версій, що дозволяють оцінювати подібність версій об’єктів

Основним недоліком алгоритму прямого порівняння в контексті виявлення порушення автентичності файлів вихідного коду було визначено те, що він не здатен оцінювати степінь схожості стрічок, тобто, якщо хоча б один елемент стрічки відрізняється, алгоритм визнає стрічки різними. При застосуванні до коду, до прикладу, перейменування змінних як одного з прийомів рефакторингу, даний метод не буде здатен виявити подібність версій файлу вихідного коду. Таким чином, є потреба шукати алгоритм, що здатен оцінювати степінь подібності стрічки коду в оригінальній версії та копії файлу.

Альтернативу неповороткому алгоритму прямого порівняння може становити один із групи алгоритмів непарного пошуку. Більшість людей сьогодні щодня взаємодіють з такими алгоритмами, коли отримують правильну відповідь у пошуковій систему на запит з орфографічною помилкою. Алгоритми непарного пошуку також набули широкої популярності у біоінформатиці, де вони використовуються для порівняння генів, хромосом та складу білків.

Для алгоритмів непарного пошуку характерна наявність метрики, яка власне і є характерною ознакою, за якою оцінюють ступінь подібності двох порівнюваних об'єктів (слів, абзаців). Сама метрика являє собою функцію відстані (різниці) між об'єктами порівняння та повинна відповідати умові нерівності трикутника:

$$p(x,y) \le p(x,z) + p(z,y), \quad x,u,z \in X, \tag{1.1}$$

де X – множина об'єктів порівняння, p – метрика. Описана функція забезпечує виконання наступної умови: сума двох правок не може бути меншою за пряму правку.

Найпопулярнішими можна назвати наступні метрики: відстань Хеммінга та відстань Левенштейна (редакційна відстань).

Відстань Хемінга має дуже обмежену зону використання. У даній моделі для зміни послідовності допускається виконання лише однієї дії – заміни. Тому, вона може застосовуватись лише для порівняння абзаців та стрічок однакової довжини, що очевидно трапляється не так часто. Для кожної пари елементів метричного простору множини слів рівної довжини визначено число, що і є відстанню Хемінга $d(x,y)$, та має відповідати основним аксіомам метрики:

1. Аксіома тотожності:

$$d(x,y) = 0 \Leftrightarrow x = y \tag{1.2}$$

2. Аксіома симетрії:

$$d(x,y) = d(y,x) \tag{1.3}$$

3. Аксіома трикутника (нерівність трикутника):

$$d(x,y) \le d(x,z) + d\,(y,z) \tag{1.4}$$

для всіх x, y і z.

Відстань Хемінга завжди має відповідати

$$d(x,y) \le n \tag{1.5}$$

де n – довжина слів у символах.

Зрозуміло, що чим більш різні слова версій, тим легше в них знайти різницю чи навпаки подібність. Для відстані Хемінга в даному контексті справедливо наступне:

1. Для того, щоб код дозволяв виявляти різницю в k (чи менше) позиціях, необхідно і достатньо, щоб найменша відстань між двома словами з різних версій коду було $\leq k + 1$.

2. Для того, щоб код дозволяв виявляти різницю в k (чи менше) позиціях, необхідно і достатньо, щоб найменша відстань між кодовими словами була $\leq 2k + 1$.

Відстань Хеммінга є не вигідним рішенням для закриття задачі виявлення подібностей у версіях файлу вихідного коду при умові застосування до копії рефакторингу, оскільки даний алгоритм не дозволить виявляти порушення автентичності у випадку наприклад додавання нових рядків чи зміни їх порядку [23].

Натомість, відстань Левенштейна передбачає значно ширший спектр операцій, її ще називають відстанню редагування. Вона так само являє собою число, що визначає мінімальну кількість операцій, які необхідно виконати над одним словом (чи стрічкою, абзацом), щоб перетворити його у інше, однак, перетворювальними операціями в даному випадку є вставлення, видалення та заміна одного символу.

Кожна операція має одиничну вагу, тобто при необхідності виконання кожної наступної операції для перетворення, до значення відстані додається одиниця. Як уже було вказано, відстань Левенштейна визначається мінімальною кількістю таких операцій, тому чим менше значення метрики, тим більша степінь подібності між двома порівнюваними послідовностями.

Як і будь-яка функція, відстань Левеншейна має властивості:

1. Завжди має бути хоча б мінімальна різниця між розмірами порівнюваних послідовностей:

$$d(S_1, S_2) \geq \big||S_1| - |S_2|\big| , \tag{1.6}$$

2. Відстань Левенштейна завжди менша за довжину довшої з двох порівнюваних послідовностей:

$$d(S_1, S_2) \leq \max(|S_1|, |S_2|) , \tag{1.7}$$

3. Відстань Левенштейна рівна нулю тоді і тільки тоді, коли порівнювані послідовності є однаковими:

$$d(S_1, S_2) = 0 \Leftrightarrow S_1 = S_2\ , \tag{1.8}$$

де $d(S_1, S_2)$ – відстань Левенштейна між стрічками S_1 та S_2, а |S| - довжина стрічки S.

Існує декілька алгоритмів, які дозволяють проводити обрахунок редакційної відстані. Деякі з них класифікуються як алгоритми динамічного програмування, що характеризує їх як такі, що спрямовані на пошук оптимального з точки зору певного критерію значення. Найпопулярнішим з них є алгоритм Вагнера-Фішера. Його часте використання обумовлене в першу чергу тим, що сам алгоритм заснований на виконанні умови нерівності трикутника, що робить його найбільш істинним для задачі пошуку редакційної відстані. Умова нерівності трикутника обмежує кількість перетворення для позиції у стрічці: жодна з них не може бути змінена більше ніж один раз. З цього випливає, що всі операції перетворення можуть виконуватись паралельно.

Алгоритм Вагнера-Фішера передбачає створення матриці редакційних відстаней між префіксами двох порівнюваних послідовностей. Відстань Левенштейна обраховується шляхом з'єднання клітинок з найменшою редакційною відстанню.

Розглянемо порядок динамічного програмування за алгоритмом Вагнера-Фішера:

1. Відбувається ініціалізація матриці, розміром (m, n), де m і n – довжина префіксу першої та другої порівнюваних послідовностей. Вимірювання редакційної відстані відбувається у комірках.

2. Матриця заповнюється з лівого верхнього до правого нижнього кута. Кожен перехід по горизонталі чи вертикалі відповідає вставці чи видаленню відповідно. Вартість кожної операції рівна одиниці.

3. Стрибок по діагоналі може вартувати одиниці, якщо символи в стовпці та у стрічці співпадають, або нулю – якщо не співпадають. Кожна комірка мінімізує вартість дії у своєму місці:

— значення комірки вище $+\ 1$ – $d[i-1, j]\ +\ 1$;

— значення комірки зліва $+ 1 - d[i, j-1] + 1$;

— значення комірки по діагоналі вліво вверх + вартість $- d[i-1, j-1] +$ вартість;

4. В результаті, число у правому нижньому кутку матриці і є відстанню Левенштейна для порівнюваних послідовностей.

Оцінка результатів динамічного програмування за алгоритмом Вагнера-Фішера відбувається за наступними принципами: найменш оптимальними вважаються результати обрахунків, при яких всі абзаци зміненої ділянки у старій версії вважаються видаленими, а всі абзаци у новій версії – доданими, тобто жодна стрічка не відповідає жодній стрічці; будь-який інший результат вважатиметься оптимальним.

У ході роботи алгоритму в більшій степені задіюються лише дві останні стрічки матриці. Це робить даний алгоритм дуже гнучким з точки зору оптимізації затрат пам’яті при обчисленнях.

З точки зору використання відстані Левенштейна, як метрики для визначення степені подібності двох версій файлу, можна виділити два досить суттєвих недоліки:

1. При перестановці місцями слів та частин слів отримують відносно велику розбіжність у визначенні значення відстані.

2. Відстань між абсолютно різними короткими словами виявляється невеликою, в той час як відстань між дуже схожими довгими словами виявляється досить значною. Тобто, важко оцінити яка з пар послідовностей є більш схожою, а яка менш [30].

Останню проблему досить легко вирішити шляхом нормалізації редакційної відстані. Цей підхід передбачає, що звичайним чином обрахована відстань Левенштейна ділиться на кількість операцій (її називають довжиною шляху редагування). Тоді ми отримуємо, так би мовити, долю редакційної відстані між порівнюваними послідовностями, яку ще називають редакційною відстанню з пост-нормалізацією.

Отже, на відміну від алгоритму прямого порівняння, який здатен лише визначати подібність стрічок на рівні «однакові»-«різні», алгоритми на основі

відстані Хемінга та відстані Левенштейна дозволяють проводити оцінку подібності версій об’єктів, що означає, що навіть за умови застосування до копії файлу вихідного коду простих прийомів рефакторингу, алгоритми дозволять виявити подібність між стрічками коду. Редакційна відстань, яку отримуємо при обрахунках за алгоритмами може служити меторикою подібності версій файлів вихідного коду. В той же час, алгоритм за відстанню Хемінга є все ж потенційно слабким в контексті розглянутої задачі, оскільки він здатен обробляти лише одну операцію – заміни. Цього на жаль недостатньо для того, щоб забезпечити належний рівень ефективності алгоритму порівняння версій навіть за умов застосування до копії простих прийомів рефакторингу, адже в арсенал зловмисника в такому випадку окрім заміни входять також операції видалення, додавання, зміни місцями тощо. Натомість алгоритм на основі відстані Левенштейна є значно результативнішим у даному контексті та може бути використаний для реалізації ефективного рішення з виявлення порушення автентичності файлу вихідного коду шляхом порівняння оригінальної версії із файлом-копією. Єдиний чинник, який не дозволяє оригінальному алгоритму Левенштейна повністю закрити потреби порівняння версій файлів вихідного коду є те, що він не покриває процедуру зміни символів місцями. Тож, існує передумова для використання алгоритму Левенштейна для здійснення обчислень редакційної відстані між кодовими послідовностями файлів вихідного коду із подальшим покращенням даного алгоритму з урахуванням специфіки предметної області та фокусом на здатність методу виявляти застосування транспозиції зі збереженням ефективності та швидкості обробки.

1.4 Критерії оцінювання успішності удосконалення алгоритму порівняння версій

Як уже було визначено раніше, алгоритм порівняння версій має забезпечувати достатню точність та об’єктивність при визначені ступеню подібності оригінальної та скопійованої версій послідовності. Оскільки розроблюваний алгоритм планується становити покращення відносно оригінального алгоритму, тож логічно випливає, що точність даного алгоритму має бути не меншою за оригінал. Оригінальний алгоритм

Левенштейна є деяким золотим стандартом у вимірі відстані між порівнюваними послідовностями, отже оцінка точності та об’єктивності удосконаленого алгоритму мусить базуватись на порівнянні результатів обчислення редакційної відстані за оригінальним та модифікованим алгоритмом. Тут важливо розуміти в яку із сторін є допустимим відхилення результатів обчислення модифікованого алгоритму відносно результатів за оригіналом. Відповідно до наведених у попередньому підрозділі висновків, недоліком алгоритму Левенштейна є те, що він не здатен виявляти застосування транспозиції до версії копії. Таким чином, додавання ще однієї операції повинно збільшити точність обчислення редакційної відстані і для випадку застосування до копії прийомів рефакторингу зменшувати редакційну відстань, тим самим вказуючи на наявність збігів між послідовностями що проходять перевірку.

Тут також постає закономірне питання про те, до якої міри є допустимим відхилення в значенні редакційної відстані обчисленому за оригінальним та модифікованим алгоритмами відповідно. В конкретно даній розглянутій предметній області на результати обчислення редакційної відстані в тому числі впливають довжини стрічки та стовпця у матриці порівнюваних послідовностей, а також степінь зміни тексту коду внаслідок застосування до нього прийомів рефакторингу, зокрема частота чи взагалі наявність застосування транспозиції як прийому рефакторингу. Тож, враховуючи те, що ми не можемо завчасно передбачити, обмежити чи контролювати жодну із вказаних умов, зафіксувати конкретну допустиму межу відхилення також неможливо. Однак, однозначно можна сказати, що результат реалізації удосконаленого алгоритму з точки зору точності та об’єктивності оцінки ступеню подібності порівнюваних послідовностей вважатиметься успішним, якщо на деякій вибірці результатів тестувань усі результати будуть рівні або менші за відповідні редакційні відстані, отримані при обчисленні за оригінальним алгоритмом.

Будь-яка зміна алгоритму впливає й на його ефективність – властивість використання обчислювальних ресурсів. Як уже було зазначено, з урахуванням особливостей предметної області, час виконання алгоритму безпосередньо впливає кількість даних які необхідно опрацювати. Отже, важливо провести оцінку часової складності алгоритму.

Часова складність є функцією довжини вхідних даних, що рівна часу роботи алгоритму при обчисленні даної вхідної послідовності. На жаль, об'єктивно оцінити час необхідний комп'ютеру для обробки алгоритму неможливо, оскільки на це впливає ще кілька факторів, крім складності самого алгоритму: швидкість процесора, тип даних, мова програмування та інші. Тож, в даному випадку прийнято нехтувати точністю та описувати складність алгоритму за асимптотичною складністю, тобто складністю, якщо розмір вхідних даних прямує до безкінечності [31].

Часову складність найчастіше виражають через нотацію Big O – відносне представлення складності алгоритму. Відносність у даному випадку означає допустимість порівняння лише подібних за типом алгоритмів. Представлення передає те, що Big O обмежується порівнянням в рамках однієї змінної (найбільш фундаментального чи запитуваного процесу). Складність пояснюється вимірюванням відносно чогось, тобто, наприклад, наскільки «складнішим» буде алгоритм на послідовності в 1 000 000 символів, відносно 10 000.

Нотація Big O одночасно описує дві характеристики алгоритму:

— часова складність – як збільшується час виконання алгоритму в залежності від розміру вхідних даних;

— просторова складність – як збільшується необхідний додатковий обсяг пам'яті необхідний для обробки алгоритму при збільшенні обсягу вхідних даних.

Big O записується як $O(f(n))$, де $f(n)$ – деяка функція, що описує як змінюється часова ефективність алгоритму при рості обсягу даних. Виділяють декілька найбільш поширених типів нотацій Big O, їх наведено у таблиці 1.2, а на рисунку 1.2 показано відповідні графіки функцій.

Таблиця 1.2 – Приклади найпоширеніших типів нотацій Big O

Назва	Позначення
Константа	$O(1)$
Логарифмічна	$O(\log n)$
Лінійна	$O\ (n)$
Логарифмічно-лінійне	$O(n \log n)$
Квадратне	$O(n^2)$
Експоненційне	$O(2^n)$
Факторіальне	$O(n!)$

Алгоритм з константною нотацією не залежить від кількості даних які необхідно опрацювати, час обробки у нього завжди сталий. Якщо алгоритм має логарифмічну нотацію, то час обробки в залежності від об’єму даних у нього збільшується ледь помітно і дуже поступово. При лінійній нотації час обробки алгоритму збільшується прямопропорційно збільшенню обсягу вхідних даних. Великий приріст у часі обробки мають алгоритми з квадратною, а ще більший із експоненційною нотацією. Найбільший приріст мають алгоритми з факторіальною нотацією: на малих об’ємах даних їх швидкість ще є прийнятною, однак на великих об’ємах даних обрахунок алгоритму починає вимагати великих ресурсів.

Варто зауважити що даний метод оцінки використання обчислювальних ресурсів має ще декілька умовностей:

1. Big O стосується лише найгіршого сценарію використання алгоритму (випадок, при якому затрати часу на виконання алгоритму для заданого розміру даних є максимальними).

2. Ігнорування констант – оскільки ми визначаємо поведінку алгоритму на масиві даних, що прямує до нескінченності, то операції з константними значеннями настільки несуттєво впливають на характер поведінки алгоритму, що ними можна знехтувати.

3. Вхідні параметри спрощуються – якщо вага якогось із елементів функції у впливі на загальний характер поведінки алгоритму є достатньо мало, щоб ним можна було знехтувати – його відкидають.

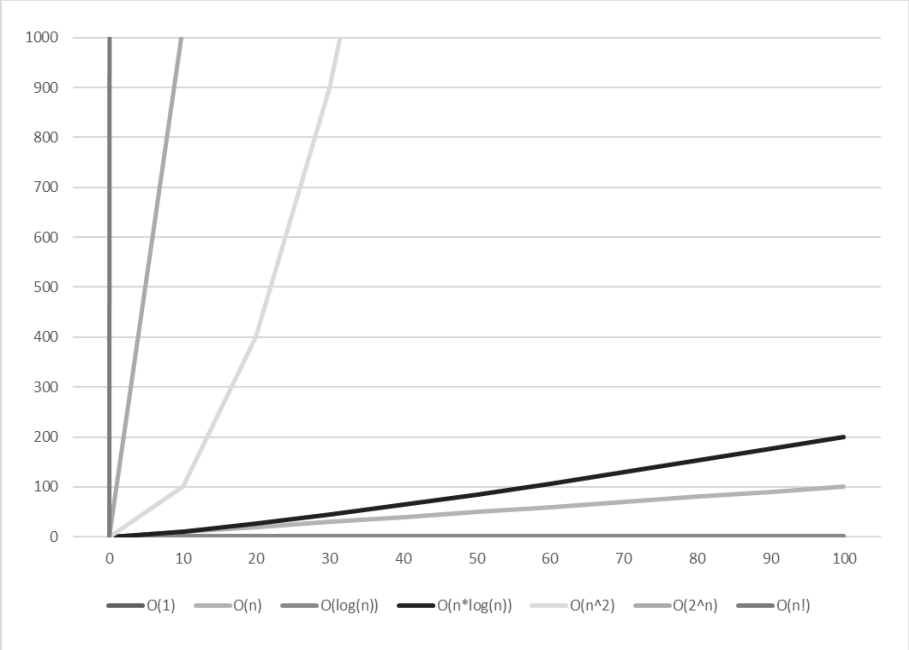


Рисунок 1.2 - Графік росту Big O для типових типів нотації

При проведенні виробничих розслідувань час відіграє інколи ключову роль, тож алгоритм мусить дозволяти проводити перевірку необхідних обсягів даних за прийнятний час [32]. Таким чином, задача при удосконаленні алгоритму буде полягати не лише у тому, аби досягнути кращих показників точності та об’єктивності обчислення редакційної відстані, але й у тому щоб зберегти чи навіть покращити часову ефективність алгоритму.

1.5 Висновки і постановка задачі

У даному розділі було проведено аналіз предметної області, а саме було досліджено сучасний стан системи захисту авторського права у сфері розробки програмного забезпечення, визначено її основні недоліки та засоби і техніки які дозволяють на сьогоднішній день їх оминати.

Також були розглянуто в чому полягає специфіка визначення авторства у сфері розробки програмного забезпечення, передачі права інтелектуальної власності та

виявлення порушення автентичності файлів вихідного коду. Виконано аналіз існуючих систем, що дозволяють проводити порівняння версій файлів, та визначено, що ті алгоритми та методи, що у них використано не здатні забезпечити необхідну об’єктивність результатів порівняння версій у контексті розслідувань випадків порушення автентичності файлі вихідного коду.

Таким чином було виявлено, що існує потреба у пошуку алгоритму, що зможе забезпечити відповідний рівень результативності та об’єктивності дослідження. Тож було виконано огляд алгоритмів, що здатні виконувати оцінку подібності файлів, в тому числі за умови застосування до файлу копії прийомів рефакторингу. Було визначено, що жоден з них не здатен забезпечити виявлення усіх навіть базових операцій рефакторингу. Отже, виявлено потребу у пошуку рішення, що базуватиметься на найефективнішому із розглянутих алгоритмів – відстані Левенштейна, однак дозволятиме виявляти подібність файлів при застосуванні до копії усіх принаймні базових операцій рефакторингу.

Було проаналізовано критерії до ефективності алгоритмів порівняння версій та визначено що ключовими для оцінки успішності розробки вдосконалення методу є похибка обчислень відносно оригінального алгоритму та часова ефективність алгоритму.

Метою даної роботи є вдосконалення методу виявлення факту порушення автентичності файлів вихідного коду на алгоритму непарного пошуку з відстанню Левенштейна в якості метрики.

Для досягнення мети дипломної роботи, необхідно виконати такі задачі:

1. Розробити удосконалений алгоритм порівняння версій на основі алгоритму Левенштейна;

2. Проаналізувати ефективність розробленого рішення;

3. Розробити програмну реалізацію підсистеми порівняння версій файлів вихідного коду на основі розробленого рішення;

4. Здійснити тестування роботи розробленої підсистеми

5. Зробити висновки щодо очікувань та реальних результатів покращення алгоритму.

2 РОЗРОБКА РІШЕННЯ ДЛЯ УДОСКОНАЛЕННЯ МЕТОДУ ВИЯВЛЕННЯ ПОРУШЕННЯ АВТЕНТИЧНОСТІ ФАЙЛІВ ВИХІДНОГО КОДУ ПРИ ЗАСТОСУВАННІ РЕФАКТОРИНГУ ДО ФАЙЛУ КОПІЇ

У даному розділі буде описано процес пошуку рішення для удосконалення алгоритму обчислення редакційної відстані за алгоритмом Левенштейна з метою покращення показників точності та об’єктивності результатів оцінки подібності порівнюваних послідовностей, при збереженні часової ефективності алгоритму. Далі буде здійснено безпосередньо проектування розробленого рішення та виконано попередню оцінку успішності реалізації запропонованого рішення.

2.1 Вибір алгоритму для удосконалення методу виявлення порушення автентичності файлів вихідного коду

Як уже було визначено у попередньому розділі, основною проблемою використання алгоритму Левенштейна для обчислення редакційної відстані стало те, що даний алгоритм покриває виявлення лише трьох дій – додавання, видалення і заміни, чого є недостатньо для того, щоб забезпечити належний рівень точності визначення ступеню подібності порівнюваних послідовностей за умови застосування до копії засобів рефакторингу. При застосуванні оригінального алгоритму Левенштейна досить значну похибку, особливо на великій послідовності коду, може внести застосування до копії коду прийому, що передбачає зміну місцями символів, слів чи цілих стрічок, тобто транспозиції.

Раніше вважалося, що враховувати транспозицію актуально лише коли ми говоримо про набір тексту і відповідно порівняння текстових послідовностей. В такому контексті ми можемо говорити про транспозицію як про модифікацію, спричинену помилками набору чи внесену навмисно перекручуванням слів у тексті з метою зміни його зовнішнього вигляду. Для коду ж транспозиція вважалась тією операцією, якою можна нехтувати, адже код як текстова послідовність має деякий набір правил, за яким він будується і навіть з метою заплутування, можливостей для транспозиції залишається досить мало. Однак, дане судження базувалось на поверхневому огляді принципів побудови коду, а також не враховує особливості мов

програмування. Транспозиція, наприклад, може знайти вияв у вигляді зміни порядку змінних у тих операціях, в яких порядок змінних не впливає на результат обчислення:

```
int a = 2;
int b = 5;
int y = a+b;
```

```
int a = 2;
int b = 5;
int y = b+a;
```

На даному прикладі ми можемо побачити, що єдина зміна яка була внесена в оригінальний код це зміна місцями змінних a і b в операції додавання, що є власне транспозицією. Однак, оригінальний алгоритм Левенштейна, не маючи у своєму арсеналі операції транспозиції оцінив би різницю між цими двома послідовностями як 2 операції заміни, замість однієї транспозиції. На даній короткій послідовності ця похибка є незначною, однак на більших відрізках коду не опрацювання транспозиції може вносити значну похибку. Таким чином, для порівняння кодових послідовностей однозначно доречним є також враховувати операцію транспозиції, як суттєвий чинник впливу на результат визначення різниці між кодовими послідовностями.

Взагалі початково, алгоритм Левенштейна розроблявся для пошуку помилок у тексті. Однак із появою комп’ютерних технологій та значного поширення клавіатурного набору постала проблема помилок набору – люди досить часто набирають замість цільового символу один з тих, що знаходяться поруч на клавіатурі, або при швидкому наборі плутають порядок символів у словах. Саме з метою додати можливість виявляти такі помилки і виник удосконалений алгоритм, що за прізвищем свого розробника дістав назву відстані Дамероу-Левенштейна, що власне є модифікацією оригінального алгоритму Левенштейна.

Основна відмінність від класичної моделі полягає у додаванні крім трьох типових дій редагування (заміна, додавання, видалення) четвертої – транспозиції. Алгоритми на основі відстані Дамероу-Левенштейна уже давно знайшли своє застосування у багатьох сферах. Першочерговим призначенням звісно було порівняння набраних шляхом клавіатурного набору текстів, зокрема у своїх дослідженнях Дамероу довів, що близько 80% помилок людина робить саме через перестановку сусідніх символів, помилку в символі чи додаванні зайвого символу. Однак, крім того, модифікований алгоритм знайшов своє застосування в

біоінформатиці та системах пошуку. Саме на основі даного алгоритму відбувається виправлення помилок вводу та пошук рекомендацій пошуку сучасними пошуковими системами.

Даний алгоритм досі не застосовувався для роботи із кодовими послідовностями, однак враховуючи виявлену проблему у роботі інших алгоритмів, саме відстань Дамероу-Левенштейна може дозволити покрити її.

Перейдемо до дослідження принципів роботи алгоритму.

Якщо позначити першу із порівнюваних стрічок через a, а другу через b, то відстань Дамероу-Левенштейна можна визначити за наступною функцією:

$$d_{a,b}(i,j) = \max(i,j), \text{якщо } \min(i,j) = 0,$$

$$\min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} \quad \text{якщо } i,j > 1 \text{ і } a_i = b_{j-1} \text{ і } a_{i-1} = b_j$$

інакше,

$$\min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \tag{2.1}$$

де $1_{(a_i \neq b_j)}$ це функція індикатор, що рівна 0 коли порівнювані послідовності збігаються, тобто $a_i = b_j$, чи 1 якщо між ними знайдено принаймні одну різницю.

Кожна з рекурсивних процедур відповідає одній із процедур модифікації тексту:

$d_{a,b}(i-1,j)+1$ – видалення символу в послідовності a з перенесенням результату в b;

$d_{a,b}(i,j-1)+1$ – додавання символу в послідовність a з перенесенням результату в b;

$d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}$ – неспівпадіння символів у послідовностях a та b;

$d_{a,b}(i-2, j-2)+1$ – перестановка двох послідовних символів послідовності a з перенесенням результату в b [33].

Порівняємо які результати отримаємо якщо обрахуємо відстані Левенштейна та Дамероу-Левенштейна для однакових пар порівнюваних стрічок символів: «bh ytn» та «b tyn». Для обрахунку редакційних відстаней використаємо алгоритм Вагнера-Фішера. Результати визначення редакційної відстані Левенштейна наведено у таблиці 2.1, а відстані Дамероу-Левенштейна – у таблиці 2.2.

Таблиця 2.1 – Результати розрахунку відстані Левенштейна для послідовностей «bh ytn» та «b tyn» за алгоритмом Вагнера-Фішера

			b	h		y	t	n
		0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
b	1	1	0	1	2	3	4	5
	2	2	1	1	1	2	3	4
t	3	3	2	2	2	2	2	3
y	4	4	3	3	3	2	3	3
n	5	5	4	4	4	3	3	3

Таблиця 2.2 – Результати розрахунку відстані Дамероу-Левенштейна для послідовностей «bh ytn» та «b tyn» за алгоритмом Вагнера-Фішера

			b	h		y	t	n
		0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
b	1	1	0	1	2	3	4	5
	2	2	1	1	1	2	3	4
t	3	3	2	2	2	2	2	3
y	4	4	3	3	3	2	2	3
n	5	5	4	4	4	3	3	2

За відстанню Левенштейна опрацьовуються лише операції видалення, додавання та заміни, отже для порівнюваних послідовностей було виявлено три операції зміни:

Операція 1 – видалення h в індексі 2

Операція 2 – видалення y в індексі 4

Операція 3 – додавання y в індексі 5

Отже, відстань Левенштейна для порівнюваних послідовностей було визначено як 3.

Натомість, за відстанню Дамероу-Левенштейна опрацьовується ще операція зміни місцями двох символів. Тож при порівнянні послідовностей було виявлено наступні операції зміни:

Операція 1 – видалення h в індексі 2

Операція 2 – транспозиція y в індексі 4 і t в індексі 5

Отже, відстань Дамероу-Левенштейна для порівнюваних послідовностей було визначено як 2.

Таким чином, можна зробити висновок, що згідно відстані Дамероу-Левенштейна різниця між порівнюваними послідовностями є меншою ніж за відстань Левенштейна, тобто вони насправді є більш подібними ніж показує оцінка за відстанню Левенштейна. Тож очевидно, що шляхом застосування відстані Дамероу-Левенштейна при порівнянні однакової пари послідовностей та застосуванні одного і того ж алгоритму обрахунку цієї відмінності можна досягнути більшої точності.

Однак зараз варто повернутись до прикладу, що був розглянутий на початку даного підрозділу:

```
int a = 2;
int b = 5;
int y = a+b;
```

```
int a = 2;
int b = 5;
int y = b+a;
```

Згідно вище зазначеного припущення, за відстанню Дамероу-Левенштейна різниця між цими двома послідовностями символів має становити 1, оскільки має місце транспозиція символів a та b вкінці третього рядка коду. Однак, згідно оригінального визначення транспозиція розглядається як перестановка двох

послідовних символів. Тож, виникає питання як буде визначатись операція зміни місцями символів, що не знаходяться один за одним. Даний нюанс був би проблемою, якби до оригінального алгоритму Левенштейна додали операцію транспозиції, однак у відстані Дамероу-Левенштейна передбачено зміну місцями символів між будь-якими індексами. Відповідні операції при заповненні матриці Вагнера-Фішера матимуть вигляд

— значення комірки вище $+ 1$ – $d[i-1,j] + 1$;

— значення комірки зліва $+ 1 - d[i,j-1] + 1$;

— значення комірки по діагоналі вліво вверх $+$ вартість $- d[i-1,j-1] +$ вартість;

— значення комірки по діагоналі вліво вверх $+$ вартість $- d[i-1-m,j-1-n] +$ вартість, де m та n – відстань між стовпцями і рядками відповідно.

Відповідно до цього проведемо обрахунки відстані Левенштейна (таблиця 2.3) та відстані Дамероу-Левенштейна (таблиця 2.4) для відрізку стрічки коду, де було застосовано транспозицію

Таблиця 2.3 – Результати розрахунку відстані Левенштейна для послідовностей «a+b» та «b+a» за алгоритмом Вагнера-Фішера

			a	+	b
		0	1	2	3
	0	0	1	2	3
b	1	1	1	2	2
+	2	2	2	1	2
a	3	3	2	2	2

Таблиця 2.4 – Результати розрахунку відстані Дамероу-Левенштейна для послідовностей «a+b» та «b+a» за алгоритмом Вагнера-Фішера

			a	+	b
		0	1	2	3
	0	0	1	2	3
b	1	1	1	2	2
+	2	2	2	1	2
a	3	3	2	2	1

За відстанню Левенштейна було виявлено дві операції зміни:

Операція 1 – заміна a в індексі 1

Операція 2 – заміна b у в індексі 3

Отже, відстань Левенштейна для порівнюваних послідовностей було визначено як 2.

Натомість, за відстанню Дамероу-Левенштейна було виявлено наступні операції зміни:

Операція 1 – транспозиція a в індексі 1 і b в індексі 3

Отже, відстань Дамероу-Левенштейна для порівнюваних послідовностей було визначено як 1.

Тож, можемо зробити висновок, що питання того чи враховується зміна місцями символів чиї індекси не є порядковими нівелюється, завдяки передбаченому відстанню Дамероу-Левенштейна варіанту розрахунку для пари транспонованих символів у не порядкових індексах. Це відповідно означає збереження точності обрахунку ступеню відмінності між порівнюваними послідовностями символів, що особливо важливо у предметній області яка розглядається у даній роботі. Якщо для звичайних текстових послідовностей більш характерними будуть транспозиція у порядкових індексах, то при порівнянні кодових послідовностей більша імовірність зустріти транспозицію у непорядкових індексах. Це спричинено тим, що на відміну від звичайного тексту, програмний код має ряд обмежень, обов’язкових елементів, позиціонування яких не можна змінювати без внесення помилок у роботу коду, тож

простір для застосування в рамках рефакторингу транспозиції з порядковими індексами звужується. Натомість ширшими є можливості для транспозиції із непорядковими індексами. Тут знову варто звернутись до прикладу із кодом для дії додавання константних змінних.

```
int y = a+b;
```

Ми не можемо змінити розташування символу «+» оскільки він може стояти лише між двома змінними, як і «=» чи «;» – їхнє розташування зумовлено їх функцією в коді.

Таким чином, було визначено, що відстань Дамероу-Левенштейна є кращим рішенням при виборі метрики оцінки відмінності між послідовностями кодів файлів вихідного коду оригіналу та копії, оскільки обчислення даної метрики передбачає операції, що дозволяють зробити оцінку відмінності більш точною та об’єктивною, а також враховувати особливості предметної області, як от транспозицію у непорядкових індексах.

2.2 Проектування рішення для удосконалення алгоритму виявлення автентичності файлів вихідного коду із застосуванням рефакторингу

Хоча використання метрики Дамероу-Левенштейна само по собі значно підвищує точність оцінки подібності вхідних послідовностей, у порівнянні із іншими подібними метриками, однак все ж тут досі не враховуються усі тонкощі предметної області, що розглядається в рамках поставленої в даній роботі задачі. Одним із таких важливих нюансів є використання літер верхнього та нижнього регістру. Якщо написання деяких невід’ємних від програмного коду виразів регламентовано правилами мови програмування, то характер назв змінних, методів тощо кожен розробник обирає самостійно: хтось використовує виключно літери нижнього регістру, хтось звик до верхнього, а хтось використовує літери обох регістрів. З одного боку, за такою незначною особливістю інколи теоретично можна відслідкувати характерний почерк конкретного розробника. Однак, в той же час зловмисник з метою приховання факту запозичення коду може просто змінити регістр у літер назв змінних і вже видати код як відмінний від оригіналу. Саме тут

проявляється основний недолік алгоритмів обрахунку метрик відмінності текстових послідовностей – вони не враховують регістр літер і сприймають одну і ту ж літеру у різних регістрах як різні символи.

Рішенням даної проблеми може стати приведення усіх символів коду до одного регістру. В такому випадку ентропія, що вноситься за рахунок використання різних регістрів для буквених символів нівелюється, а алгоритм враховує лише реальний зміст кодової послідовності та порядок його подання.

Наступним питання, що потребує вирішення відповідно до поставленої у даній роботі задачі є питання часу виконання задачі порівняння вхідних даних алгоритмом. При проведенні реальних службових розслідувань факту несанкціонованого копіювання вихідного коду із порушенням авторських прав, для того щоб провести повноцінну перевірку, в рамках слідчих дій доведеться порівняти між собою великі масиви даних оригінальних файлів із їх можливими копіями. Однак, часові ресурси такого розслідування майже завжди є обмеженими та не терплять зволікань, тож надзвичайно важливо мати засоби, що дозволять проводити дану перевірку із якомога меншими часовими затратами, або якщо говорити точніше, то які володітимуть найвищою продуктивністю сканування порівнюваних послідовностей за одиницю часу. Саме тому важливо щоб реалізація алгоритму оцінки відстані Дамероу-Левенштейна дозволяла максимально оптимізувати час, що затрачається на сканування та порівняння вхідних даних.

Якщо повернутись до особливостей предметної області, що розглядається в рамках поставленої задачі, тут можна виділити один важливий аспект, що можна використати саме на користь оптимізації часу обробки даних алгоритмом. Як уже було зазначено, код хоч і розглядається алгоритмом як звичайна текстова послідовність, однак за своєю суттю має деякі закономірності. Незалежно від того якою мовою програмування було написано код, у кожній з них є набір виразів та символів, що є обов’язковими і які не підлягають зміні, навіть під дією рефакторингу, оскільки їх зміна неминуче призведе до втрати кодом працездатності. Прикладом таких виразів є назви змінних, назви циклів, символи відкриття і закриття циклу, заголовки конструкції обробки виключень тощо. Дані вирази з точки зору алгоритму

порівняння так само є послідовністю символів, однак такою, що завжди буде однаковою і в оригінальній версії і у коді копії, оскільки своїм формулюванням підпорядковуються правилам заданим мовою програмування, якою було написано код програми. Отже, незалежно від того якої міри зміненою шляхом рефакторингу є послідовність копії відносно оригіналу, ці вирази завжди будуть однаковими. При цьому, саме такі вирази становлять більшу частину кодової послідовності, в той же час фактично несучи меншість впливових ресурсів на результати порівняння. Таким чином, обробка цих відрізків коду вимагає використання надлишкових ресурсів, як результат збільшуючи час обробки вхідних даних алгоритмом.

Рішенням даної задачі може стати виключення ряду таких виразів із загальної послідовності символів коду ще на підготовчому етапі, перед початком обробки вхідних послідовностей алгоритмом. Таким чином алгоритм отримує в обробку уже значно коротшу у порівнянні із початковою версією послідовність, до того ж лише таку у якій доля імовірності збігів є значно меншою, тобто алгоритм не витрачає час на пошук відмінностей у тих частинах послідовностей де їх заздалегідь немає.

Таким чином, перед початком порівняння версій коду та обчислення редакційної відстані, кожна із послідовностей має пройти двоступеневу підготовку:

1. Переведення символів до одного регістру.

2. Видалення з послідовності усіх повторюваних, невід’ємних від коду символьних послідовностей та виразів, відповідно до мови програмування, на якій написано код.

Процес виконання даної підготовки кодової послідовності показано на рисунку Б1 у додатку Б.

Розглянемо алгоритм виконання даної підготовки детальніше:

Крок 1. Завантаження порівнюваних кодових послідовностей.

Крок 2. Приведення буквених символів завантажених послідовностей до верхнього регістру.

Крок 3. Очищення кодових послідовностей від невід’ємних виразів коду, шляхом заміни відповідних визначених наборів символів на null.

Крок 4. Вивід очищених версій порівнюваних послідовностей.

Крок 5. Порівняння послідовностей через обрахунок відстані Дамероу-Левенштейна за алгоритмом Вагнера-Фішера.

Крок 6. Вивід результату порівняння – відстані Дамероу-Левенштейна між порівнюваними послідовностями.

Отже, виявлено, що алгоритм визначення відстані Дамероу-Левенштейна, через те що початково був розроблений для порівняння текстових послідовностей не враховує регістр буквених символів, та визначає однакові літери у різних регістрах як різні символи. Однак, для предметної області що розглядається у даній роботі даний фактор є чинником що призводить до внесення похибки у результати обрахунку редакційної відстані, адже у кодовій послідовності регістр літер відіграє значно меншу роль у зміні змісту коду, у порівняння із звичайним текстом. Таким чином, для досягнення більшої точності обрахунку відстані Дамероу-Левенштейна між двома послідовностями вихідного коду пропонується додати операцію виведення усіх символів обох послідовностей до одного регістру перед початком їх порівняння за алгоритмом Вагнера-Фішера.

Також, з метою покращення показників швидкості проведення перевірки подібності запропонованим алгоритмом, пропонується проводити виключення усіх повторюваних та заздалегідь однакових елементів послідовностей, що обумовлено особливістю коду як текстової послідовності – незалежно від мови програмування якою написано код, у ньому завжди присутні деякі вирази, які не підлягають зміні навіть під час модифікації методами рефакторингу, оскільки це призведе до втрати кодом працездатності. Таким чином, дані послідовності варто піддати виключенню перед початком проведення порівняння послідовностей коду, тим самим скоротивши обсяг вхідних даних, без втрати точності порівняння.

2.3 Попередня оцінка успішності запропонованого рішення для удосконалення алгоритму порівняння версій

Як уже було зазначено у 1 розділі, показниками успішності реалізації запропонованого рішення є:

1. Відношення результату обрахунку редакційної відстані за удосконаленою версією відстані Дамероу-Левенштейна до результатів отриманих при порівнянні тих же зразків коду за відстанню Левенштейна, як показник покращення точності визначення редакційної відстані.

2. Покращення показників швидкості обробки удосконаленим алгоритмом порівняння версій коду за відстанню Дамероу-Левенштейна у порівнянні зі швидкістю обробки порівняння за відстанню Левенштейна.

Показник покращення точності досить легко отримати за рахунок обчислення відсоткового відношення між результатом отриманим за удосконаленим алгоритмом для визначення відстані Дамероу-Левенштейна до результату відстані Левенштейна:

$$R = \frac{D_{DLm}}{D_L} * 100\% \quad (2.2)$$

Де, R – відсоткове відношення різниці між результатами обчислень відстані Дамероу-Левенштейна за удосконаленим алгоритмом до відстані Левенштейна;

D_{DL_m}– відстань Дамероу-Левенштейна, обчислена за удосконаленим алгоритмом;

D_L– відстань Левенштейна.

Крім того, варто здійснити обчислення відстані Дамероу-Левенштейна за оригінальним алгоритмом та порівняти результати оригінального та модифікованого алгоритмів:

$$R = \frac{D_{DL}}{D_L} * 100\% \quad (2.3)$$

Де, D_{DL}– відстань Дамероу-Левенштейна, обчислена за оригінальним алгоритмом.

Відсоткові відношення, отримані при обрахунку відстані Дамероу-Левенштейна за оригінальним та удосконаленим алгоритмами відносно відстані Левенштейна мають бути однаковими.

Виконаємо тестове обчислення для деякої стрічки коду та її зміненої копії

int origOps = Convert.ToInt32(label8.Text) – оригінал

int oriogps = Convert.ToInt32(Label8.Text) – копія

На рисунку 2.1 наведено результати обрахунків відстані Левенштейна, а на рисунку 2.2 – відстані Дамероу-Левенштейна за оригінальним алгоритмом.

			i	n	t		o	r	i	g	O	p	s		=		C	o	n	v	e	r	t	.	T	o	l	n	t	3	2	(	l	a	b	e	l	8	.	T	e	x	t	)
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
i	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
n	2	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
t	3	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	4	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
o	5	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
r	6	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
i	7	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
o	8	8	7	6	5	4	3	2	1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
g	9	9	8	7	6	5	4	3	2	1	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
p	10	10	9	8	7	6	5	4	3	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
s	11	11	10	9	8	7	6	5	4	3	3	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	12	12	11	10	9	8	7	6	5	4	4	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
=	13	13	12	11	10	9	8	7	6	5	5	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	14	14	13	12	11	10	9	8	7	6	6	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
C	15	15	14	13	12	11	10	9	8	7	7	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
o	16	16	15	14	13	12	11	10	9	8	7	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
n	17	17	16	15	14	13	12	11	10	9	8	8	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
v	18	18	17	16	15	14	13	12	11	10	9	9	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
e	19	19	18	17	16	15	14	13	12	11	10	10	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
r	20	20	19	18	17	16	15	14	13	12	11	11	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
t	21	21	20	19	18	17	16	15	14	13	12	12	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
.	22	22	21	20	19	18	17	16	15	14	13	13	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	23	23	22	21	20	19	18	17	16	15	14	14	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
o	24	24	23	22	21	20	19	18	17	16	15	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
l	25	25	24	23	22	21	20	19	18	17	16	16	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
n	26	26	25	24	23	22	21	20	19	18	17	17	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
t	27	27	26	25	24	23	22	21	20	19	18	18	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	28	28	27	26	25	24	23	22	21	20	19	19	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	29	29	28	27	26	25	24	23	22	21	20	20	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(	30	30	29	28	27	26	25	24	23	22	21	21	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14
L	31	31	30	29	28	27	26	25	24	23	22	22	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3	4	5	6	6	7	8	9	10	11	12	13
a	32	32	31	30	29	28	27	26	25	24	23	23	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	3	4	5	6	7	8	9	10	11	12	13
b	33	33	32	31	30	29	28	27	26	25	24	24	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	5	4	3	4	5	6	7	8	9	10	11	12
e	34	34	33	32	31	30	29	28	27	26	25	25	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	6	5	4	3	4	5	6	7	8	9	10	11
l	35	35	34	33	32	31	30	29	28	27	26	26	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	6	5	4	3	4	5	6	7	8	9	10
8	36	36	35	34	33	32	31	30	29	28	27	27	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	7	6	5	4	3	4	5	6	7	8	9
.	37	37	36	35	34	33	32	31	30	29	28	28	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	8	7	6	5	4	3	4	5	6	7	8
T	38	38	37	36	35	34	33	32	31	30	29	29	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	9	8	7	6	5	4	3	4	5	6	7
e	39	39	38	37	36	35	34	33	32	31	30	30	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	10	9	8	7	6	5	4	3	4	5	6
x	40	40	39	38	37	36	35	34	33	32	31	31	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	11	10	9	8	7	6	5	4	3	4	5
t	41	41	40	39	38	37	36	35	34	33	32	32	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	12	11	10	9	8	7	6	5	4	3	4
)	42	42	41	40	39	38	37	36	35	34	33	33	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	13	12	11	10	9	8	7	6	5	4	3

Рисунок 2.2 – Результат обрахунку відстані Левенштейна для порівнюваних стрічок

			i	n	t		o	r	i	g	O	p	s		=		C	o	n	v	e	r	t	.	T	o	l	n	t	3	2	(	l	a	b	e	l	8	.	T	e	x	t	)
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
i	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
n	2	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
t	3	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	4	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
o	5	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
r	6	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
i	7	7	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
o	8	8	7	6	5	4	3	2	1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
g	9	9	8	7	6	5	4	3	2	1	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
p	10	10	9	8	7	6	5	4	3	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
s	11	11	10	9	8	7	6	5	4	3	3	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	12	12	11	10	9	8	7	6	5	4	4	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
=	13	13	12	11	10	9	8	7	6	5	5	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	14	14	13	12	11	10	9	8	7	6	6	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
C	15	15	14	13	12	11	10	9	8	7	7	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
o	16	16	15	14	13	12	11	10	9	8	7	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
n	17	17	16	15	14	13	12	11	10	9	8	8	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
v	18	18	17	16	15	14	13	12	11	10	9	9	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
e	19	19	18	17	16	15	14	13	12	11	10	10	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
r	20	20	19	18	17	16	15	14	13	12	11	11	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
t	21	21	20	19	18	17	16	15	14	13	12	12	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
.	22	22	21	20	19	18	17	16	15	14	13	13	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	23	23	22	21	20	19	18	17	16	15	14	14	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
o	24	24	23	22	21	20	19	18	17	16	15	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
l	25	25	24	23	22	21	20	19	18	17	16	16	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
n	26	26	25	24	23	22	21	20	19	18	17	17	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
t	27	27	26	25	24	23	22	21	20	19	18	18	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	28	28	27	26	25	24	23	22	21	20	19	19	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	29	29	28	27	26	25	24	23	22	21	20	20	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(	30	30	29	28	27	26	25	24	23	22	21	21	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14
L	31	31	30	29	28	27	26	25	24	23	22	22	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3	4	5	6	6	7	8	9	10	11	12	13
a	32	32	31	30	29	28	27	26	25	24	23	23	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	3	4	5	6	7	8	9	10	11	12	13
b	33	33	32	31	30	29	28	27	26	25	24	24	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	5	4	3	4	5	6	7	8	9	10	11	12
e	34	34	33	32	31	30	29	28	27	26	25	25	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	6	5	4	3	4	5	6	7	8	9	10	11
l	35	35	34	33	32	31	30	29	28	27	26	26	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	6	5	4	3	4	5	6	7	8	9	10
8	36	36	35	34	33	32	31	30	29	28	27	27	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	7	6	5	4	3	4	5	6	7	8	9
.	37	37	36	35	34	33	32	31	30	29	28	28	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	8	7	6	5	4	3	4	5	6	7	8
T	38	38	37	36	35	34	33	32	31	30	29	29	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	9	8	7	6	5	4	3	4	5	6	7
e	39	39	38	37	36	35	34	33	32	31	30	30	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	10	9	8	7	6	5	4	3	4	5	6
x	40	40	39	38	37	36	35	34	33	32	31	31	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	11	10	9	8	7	6	5	4	3	4	5
t	41	41	40	39	38	37	36	35	34	33	32	32	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	12	11	10	9	8	7	6	5	4	3	4
)	42	42	41	40	39	38	37	36	35	34	33	33	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	13	12	11	10	9	8	7	6	5	4	3

Рисунок 2.3 – Результат обрахунку відстані Дамероу-Левенштейна для порівнюваних стрічок

Порівняння за відстанню Левенштейна визначає різницю між цими стрічками як $D_L = 3$:

1. Заміна символу «g» на «o».

2. Заміна символу «O» на «g».

3. Заміна символу «L» на «l».

Порівняння за відстанню Дамероу-Левенштейна також дало результат D_{DL}=3:

1. Заміна символу «g» на «o».

2. Заміна символу «O» на «g».

3. Заміна символу «L» на «l».

Символи «O», «o» та «L», «l» сприймаються оригінальним алгоритмом як різні символи, тож зміну у слові «origOps»-«oriogps» було ідентифіковано не як транспозицію, а як дві заміни символів.

Тепер виконаємо таке ж обчислення відстані Дамероу-Левенштейна, тільки цього разу згідно запропонованого удосконаленого алгоритму.

Першою дією приводимо усі буквені символи обох стрічок до верхнього регістру:

INT ORIGOPS = CONVERT.TOINT32(LABEL8.TEXT) – оригінал

INT ORIOGPS = CONVERT.TOINT32(LABEL8.TEXT) – копія

Тепер видаляємо усі символи та їх комбінації, що складовими стандартних для коду даної мови програмування виразів. У даному випадку це «INT», «=», «CONVERT.TO.INT32», «(», «.», «TEXT», «)», « ». В результаті для проведення перевірки залишаються наступні стрічки:

ORIGOPSLABEL8 – оригінал

ORIOGPSLABEL8 – копія

На рисунку 2.4 показано результати обрахунку відстані Дамероу-Левенштейна за удосконаленим алгоритмом.

			O	R	I	G	O	P	S	L	A	B	E	L	8
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13
O	1	1	0	1	2	3	4	5	6	7	8	9	10	11	12
R	2	2	1	0	1	2	3	4	5	6	7	8	9	10	11
I	3	3	2	1	0	1	2	3	4	5	6	7	8	9	10
O	4	4	3	2	1	1	1	2	3	4	5	6	7	8	9
G	5	5	4	3	2	1	1	2	3	4	5	6	7	8	9
P	6	6	5	4	3	2	2	1	2	3	4	5	6	7	8
S	7	7	6	5	4	3	3	2	1	2	3	4	5	6	7
L	8	8	7	6	5	4	4	3	2	1	2	3	4	5	6
A	9	9	8	7	6	5	5	4	3	2	1	2	3	4	5
B	10	10	9	8	7	6	6	5	4	3	2	1	2	3	4
E	11	11	10	9	8	7	7	6	5	4	3	2	1	2	3
L	12	12	11	10	9	8	8	7	6	5	4	3	2	1	2
8	13	13	12	11	10	9	9	8	7	6	5	4	3	2	1

Рисунок 2.4 – Результат обрахунку відстані Дамероу-Левенштейна для порівнюваних стрічок за удосконаленим алгоритмом

Порівняння по удосконаленому алгоритму за відстанню Дамероу-Левенштейна дало результатD_{DL_m}=1:

1. Транспозиція символів «G» на «O».

Таким чином бачимо, що завдяки застосуванню удосконаленому алгоритму вдалося точніше визначити відстань Дамероу-Левенштейна між тестовими стрічками коду та довести що вони є майже однаковими. Точність обчислення при цьому зросла на:

$$R = \frac{1}{3} * 100\% = 0{,}333 * 100\% \approx 33\%$$

Однак, такий приріст точності характерний лише для конкретно даного розглянутого випадку (даної комбінації змін, даної довжини стрічок тощо). Дане значення є динамічним і залежить від довжини порівнюваних послідовностей коду та ступеню зміни копії відносно оригіналу. Тому, щоб отримати більш точну картину приросту точності обчислень потрібно провести тестування на вибірці кодових послідовностей, з різною довжиною, але довжина цих послідовностей має бути значно більшою за ту, що були використані для ручного тестування вище.

Аналогічно, довжина послідовностей які порівнюються безпосередньо впливає і на швидкість роботи алгоритму. Визначимо Big O для алгоритму Вагнера-Фішера для обчислення відстані Дамероу-Левенштейна. Для цього розглянемо псевдокод даного алгоритму:

```
int levensteinInstruction(String s1, String s2, int InsertCost, int DeleteCost, int ReplaceCost):
 D[0][0] = 0
 for j = 1 to N
    D[0][j] = D[0][j - 1] + InsertCost
 for i = 1 to M
   D[i][0] = D[i - 1][0] + DeleteCost
 for j = 1 to N
   if S1[i] != S2[j]
     D[i][j] = min(D[i - 1][j] + DeleteCost,
     D[i][j - 1] + InsertCost,
     D[i - 1][j - 1] + ReplaceCost
     D[i - 2][j - 2] + TranspositonCost)
 else
   D[i][j] = D[i - 1][j - 1]
 return D[M][N]
```

Звідси видно, що основне навантаження алгоритму полягає у вкладеному циклі, де внутрішній відбувається до значення N, а зовнішній – до M, де M та N – довжини стрічок тексту що порівнюються. Отже, для алгоритму Вагнера-Фішера при визначенні відстані Дамероу-Левенштейна $O(m * n)$. Таку ж часову ефективність має алгоритм Вагнера-Фішера при визначенні відстані Левенштейна.

Для спрощення розуміння, $O(m * n)$ прирівнюють до квадратної нотації $O(m * n) = O(n^2)$. Отже, графік часової ефективності алгоритму Вагнера-Фішера матиме параболічну форму (див. рис. 1.1). Однак варто розуміти, що m та n не завжди будуть рівними, навіть навпаки – у більшості випадків вони будуть різними, тож графік для кожної із пар порівнюваних кодових послідовностей буде різним. Однозначно можна сказати, що він завжди буде більш пологий ніж графік $O(n^2)$, за умови що n – довша із порівнюваних послідовностей.

Розроблений удосконалений алгоритм передбачає скорочення довжини послідовностей коду, що перевіряються ще перед початком роботи алгоритму порівняння, отже автоматично навантаження на алгоритм у порівнянні із оригінальним рішенням буде зменшуватись (рисунок 2.5).

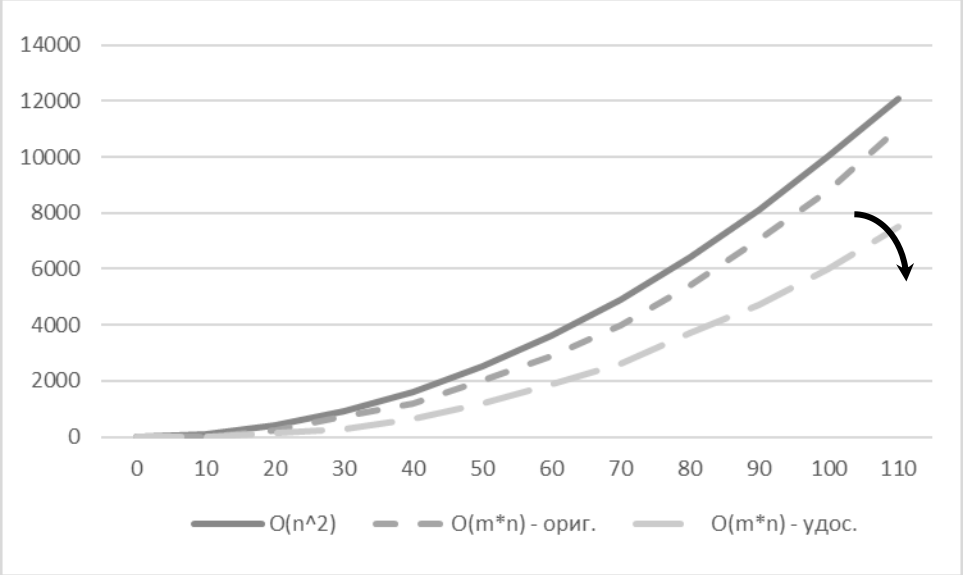


Рисунок 2.5 – Графік очікуваної поведінки нотації Big O для удосконаленого алгоритму у порівнянні із оригінальними

Отже, при проведенні тестування очікуваний характер поведінки зміни часової ефективності удосконаленого алгоритму відносно оригінального матиме вигляд як показано на рисунку 2.5: парабола удосконаленого алгоритму буде більш пологою одночасно ніж парабола $O(n^2)$ та параболо-видний графік $O(m * n)$ оригінального алгоритму.

2.4 Висновки до розділу 2

У даному розділі було розглянуто можливість використання алгоритму для визначення відстані Дамероу-Левенштейна як альтернативи відстані Левенштейна для визначення відмінності при порівнянні файлів вихідного коду програм. Було виявлено, що відстань Дамероу-Левенштейна здатна виявити застосування транспозиції як операції зміни вигляду тексту коду, що значно підвищує точність обчислені для предметної області що розглядається.

Також, було розглянуто можливість усунення деяких недоліків алгоритму в контексті даної предметної області, а саме врахування регістру літер, а також необхідність перевіряти повторювані та невід’ємні від коду послідовності символів, що заздалегідь не впливають на результат перевірки. Відповідно було запропоновано рішення для кожного із виявлених недоліків: приведення усіх буквених символів обох

порівнюваних послідовностей до одного регістру та виключення усіх повторюваних та невід’ємних від коду виразів перед виконанням перевірки за алгоритмом Вагнера-Фішера. Додатково було проведено тестування запропонованого рішення шляхом порівняння результатів обрахунку відстані Левенштейна та відстані Дамероу-Левенштейна за оригінальним та удосконаленим алгоритмом. Було визначено, що для того щоб зробити більш точні висновки про успішність реалізації удосконалення до алгоритму необхідно провести тестування рішення на вибірці із кодових послідовностей різної довжини. Отже, існує потреба розробки програмного рішення, яке дозволило б здійснити багатократну перевірку роботи удосконаленого алгоритму на кодових послідовностях різної довжини та порівняти показники точності та швидкості роботи алгоритму із відповідними показниками оригінального алгоритму визначення відстані Левенштейна, як загальноприйнятого стандарту визначення редакційної відстані.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ВИЯВЛЕННЯ АВТЕНТИЧНОСТІ ФАЙЛІВ ВИХІДНОГО КОДУ НА ОСНОВІ РОЗРОБЛЕНОГО УДОСКОНАЛЕНОГО МЕТОДУ

У даному розділі буде виконано програмну реалізацію розробленого удосконаленого алгоритму обчислення відстані Дамероу-Левенштейна для порівняння файлів вихідного коду програм. Для цього першочергово буде виконано вибір засобів розробки. Далі буде безпосередньо виконано програмну реалізацію оригінального та удосконаленого алгоритму. За допомогою програмної реалізації буде проведено тестування алгоритмів на деякій вибірці файлів вихідного коду програм з метою визначення ефективності розробленого рішення з точки зору покращення точності обчислень та швидкості обробки алгоритму та відповідно оцінки успішності удосконалення алгоритму.

3.1 Вибір засобів програмування та середовища розробки

Виходячи з поставленої задачі та алгоритму роботи програми, обраних механізмів реалізації функціоналу підсистеми, було здійснено вибір мови та засобів програмування і розробки.

Розроблювана підсистема передбачає обробку великого обсягу даних, підтримку бази даних та роботу декількох математично непростих алгоритмів. Оскільки задачею системи є забезпечення безпеки даних, що зберігаються в базі даних та обробляються у модулях підсистеми, засоби розробки повинні надавати можливості для реалізації функцій безпеки та безпосередньо забезпечувати безпеку коду підсистеми під час її розробки. Також, так як розроблювані модулі становлять програмну підсистему, яка може бути в подальшому інтегрована у певну корпоративні систему, варто передбачити можливість інтеграції коду підсистеми з будь-яким іншим кодом. Уся мета роботи підсистеми розгортається навколо файлів вихідного коду, які є основним об’єктом інтересів підсистеми.

Зважаючи на описані критерії, при виборі парадигми програмування вибір одразу падає на об’єктно-орієнтоване програмування. Об’єктно-орієнтоване програмування, на відміну від структурної парадигми, зосереджується не просто на

послідовності дій та логіці, а представляє програму як систему взаємодіючих між собою об’єктів. Це особливо актуально для розроблюваної підсистеми захисту, оскільки алгоритм її є досить розгалуженим: порядок дій підсистеми не є фіксованим, а змінюється залежно від дій користувача. Це відповідає одному з принципів об’єктно-орієнтованого програмування, який передбачає виконання усіх дій та розрахунків шляхом взаємодії між об’єктами. Тобто, один об’єкт потребує для виконання власної дії, результату виконання дії іншого об’єкта. Алгоритм вимагає роботи із основним об’єктом – файлом виконуваного коду та його властивостями. Усі математичні алгоритми також працюють і створюють значення властивостей для цього об’єкта.

Використання принципів об’єктно-орієнтованого програмування дозволить значно спростити розробку програмної підсистеми та подальший процес налагоджування та пошуку помилок. В першу чергу, це пов’язано зі спрощенням роботи з великими об’ємами даних, а все завдяки наслідуванню. Даний принцип забезпечується наявністю системи класів, у яких задаються функції, що є спільними для ряду об’єктів. Ці об’єкти стають екземплярами класу і наслідують ці властивості. Тобто, зникає потреба описувати однакові властивості для кожного об’єкта. До того ж, наслідування передбачається і між класами, тобто один клас може сам бути підкласом іншого і наслідувати від нього властивості та функції. Принцип інкапсуляції дозволяє забезпечити виконання базових принципів безпеки ще на рівні коду. Він передбачає приховування деталей роботи класів від об’єктів, що є їх екземплярами чи використовують їх. З точки зору розмежування доступу, об’єктно-орієнтоване програмування надає також додаткові можливості за рахунок поліморфізму. Ця властивість дозволяє змусити різні класи об’єктів реагувати по різному на однакові команди.

Об’єктно-орієнтованими є більшість сучасних мов програмування: Java, C#, C++, Python, Ruby та ін. Серед них варто виділити C#. Вона вирізняється одночасною простотою та силою, завдяки якій дозволяє програмістам розробляти додатки різного рівня складності, функціоналу та сфери діяльності. Варто відзначити важливі функції C#, які є надзвичайно корисними та зручними у контексті розробки підсистеми

захисту. В першу чергу це швидкість роботи додатків та програм, які написані даною мовою: вона належить до мов компільованого типу, за рахунок чого досягається достатньо висока швидкість виконання програм. Ще однією перевагою C# є висока надійність роботи додатків та програм, що написані даною мовою програмування. Це досягається завдяки роботі виконуваного середовища CLR (Common Language Runtime). CLR запускає файл програми у віртуальному процесі, за рахунок чого забезпечується безпека виконання, адже при виникненні будь-яких помилок, вони не нестимуть впливу на роботу самої системи. Окрім безпечного виконання, CLR підтримує такі важливі служби як управління пам'яттю, управління потоками, обробка винятків та збір сміття, які також є запорукою надійності виконуваного коду. CLR звільняє розробника від необхідності піклуватись про постійне очищення пам'яті, оскільки вона звільняється автоматично коли об'єкт стає недоступним та невикористовуваним. Обробка винятків виконується завдяки наявності типів, що допускають значення Null. Таким чином, виникає автоматичний захист від появи змінних, що не містять посилань на визначені об'єкти системи. Управління потоками досягається завдяки гнучкому синтаксису LINQ, який описує загальні принципи для обробки даних з будь-якого джерела. Також він надає можливості для створення розподілених систем в яких переважають асинхронні операції, що в свою чергу є ще однією запорукою швидкості роботи додатків та програм на C#. З точки зору розробника, C# - це проста в освоєнні мова програмування, з безліччю синтаксичних конструкцій та можливостей роботи, що дозволяють проводити максимальну оптимізацію та спрощення коду, а отже скорочувати кількість помилок у коді та зменшувати імовірність помилок виконання [36]. Не останню роль у цьому відіграє платформа .Net, що підтримує C#.

.Net – кероване середовище виконання, що надає ряд функцій та служб для виконання програм і додатків. Платформа включає в себе уже згадуване середовище CLR та бібліотеку класів .Net Framework – там наведені шаблони перевіреного коду, який розробники можуть викликати з розроблюваних додатків. C# як мова програмування розроблялась спеціально для розробки в рамках платформи .Net, однак платформа підтримує одразу декілька мов: C++, Delphi та ін. .Net передбачає

функцію взаємодії між мовами. Компілятори мов, орієнтованих на роботу на базі .Net, видають код, який називають CIL (Common Intermediate Language), тобто проміжний код. Він згодом компілюється уже інструментами CLR. Система типів є однаковою для усіх мов програмування .Net Framework, а усі базові типи унаслідуються із цієї системи. Таким чином, .Net дозволяє проводити інтеграцію частин додатків написаних різними мовами в одну програмну систему, що є надзвичайно актуальним з точки зору розроблюваної системи захисту. Не менш важливою особливістю платформи є кросплатформенність. На сьогоднішній день, останні версії платформи підтримуються на більшості популярних технологій та платформ: Windows, MacOS, Linux, Android, iOS тощо. Тобто, засобами .Net Framework можна розробляти додатки мовою C# для будь-якої з цих платформ. З точки зору розробника, .Net надає значні спрощення роботи, а все завдяки великій бібліотеці класів. Користуючись стандартними бібліотеками типів та членів, розробник позбавляється необхідності описувати самостійно стандартні низько рівневі операції. Таким чином, без бібліотеки класів не обходиться розробка жодного виду програми.

Розроблювана підсистема захисту потребує наявності простого у користуванні та інтуїтивно зрозумілого користувацького інтерфейсу. Тому при виборі технології розробки звертається увага на ці критерії, а також на наявність та можливості роботи інструментів відображення великих обсягів даних, надійність інструментів взаємодії з базами даних [37]. З цієї точки зору варто звернути увагу на Windows Forms.

Windows Forms – це технологія .Net Framework, яка дозволяє розробляти додатки з повнофункціональним графічним користувацьким інтерфейсом – інтелектуальні клієнти. Вони є простими з точки зору оновлення та розгортання на різних платформах та забезпечують безпечний доступ до ресурсів системи.

Як технологія Windows Forms є набором бібліотек управління, які містять шаблони, що спрощують виконання простих стандартних функцій об’єктів. Кожен об’єкт у Windows Forms - це елемент управління, оскільки він відповідає за виконання певних маніпуляцій з даними в програмній системі. Усі об’єкти розміщуються на спеціальній поверхні – формі. Таким чином, можна сказати що основний принцип написання додатку на Windows Forms – це розміщення елементів

управління в межах форми та прописування команд, згідно яких об’єкт управління має реагувати на конкретну дію користувача чи зміну в системі. Для зручної роботи з елементами управління розробник завжди може скористатись зручним інструментом у складі Windows Forms – конструктором. В рамках цього модуля розробник може працювати з реальними видимими об’єктами управління, додавати, видаляти, переміщувати їх по формі, а також автоматично створювати обробники подій – методи, що описують дії, які має виконати програма після вчинення певної дії з конкретним об’єктом управління з боку користувача чи іншої зміни в системі. Наявність цього інструмента значно спрощує та пришвидшує процес роботи над написанням програми, адже розробнику не потрібно вручну описувати ініціалізацію кожного об’єкта, його параметри, розташування тощо. Цей код генерується автоматично.

Найбільш важливим з точки зору створення розроблюваної системи захисту є наявність гнучких інструментів обробки та відображення даних. У Windows Forms це можливо завдяки наявності інструмента BindingSource. Цей компонент виконує з’єднання з джерелом даних і через елемент управління Binding Navigator створює простий компонент для роботи із записами. Бібліотека BindingSource включає методи для прив’язування даних до елементів управління, зміни записів і збереження змін у вихідному джерелі, переходу між записами. Серед приємних доповнень до усіх функцій роботи з даними у Windows Forms можна виокремити можливість зберігати дані про користувацькі звички при користуванні програмою або додатком. Наприклад, останні введені положення у пошуку чи постійне місце зберігання файлів тощо. Для цього використовують прив’язку даних через параметри. Параметри звичок користувача визначають редактором коду і зберігаються у форматі XML на користувацькому комп’ютері. При наступному виконання програми, цей файл автоматично зчитується у пам'ять програми.

Для зручного відображення даних з джерела, у Windows Forms передбачений елемент управління DataGridView. Цей інструмент дозволяє відображати дані з джерела у звичному для користувача табличному поданні. А завдяки можливості користування конструктором форм, розробнику не доводиться описувати окремо

деякі доступні налаштування таблиці: фіксацію стрічок та стовпців, відображення елементів управління всередині таблиці та навіть налаштування окремих комірок таблиці [38].

Таким чином, завдяки використанню Windows Forms в рамках платформи .Net значно пришвидшується та спрощується робота розробника. Автоматизація роботи з елементами управління, наявність готової бібліотеки базових класів та інструментів роботи з джерелом даних дає можливість зосередитись на розробці головних алгоритмів роботи програми та її безпеки, не витрачаючи час та ресурси уваги на пропрацьовування стандартних типових елементів.

При створенні додатків та програм на C# є декілька варіантів для вибору середовища розробки: ProjectRider, Eclipse, MonoDevelop, Code::Blocks, Visual Studio та інші. Кожне з них має свої переваги та недоліки, зважаючи на які розробники обирають для себе оптимальний варіант за функціоналом, ресурсами та відповідністю поставленій задачі. Для створення розроблюваної системи захисту було зроблено вибір у сторону Visual Studio.

Перш за все, такий вибір пов'язаний з тим, що дане середовище розробки, як і сама мова програмування C# та платформа .Net є офіційними розробками компанії Microsoft. А отже, Visual Studio містить увесь набір можливостей та функцій, необхідних для розробки і першою отримує усі оновлення, пов’язані зі змінами у платформі чи оновленнями мови програмування.

Функціональне наповнення цього інтегрованого середовища розробки є значно ширший ніж у аналогів. Окрім редагування та відладжування коду, у Visual Studio присутні інструменти компіляції, графічні конструктори, засоби автозавершення коду та багато інших. Розробнику доступний цілий набір функцій, що підвищують продуктивність при створенні коду програми. Серед найбільш важливих слід відзначити підкреслення помилок, автоматичне доповнення коду та засоби рефакторингу.

Visual Studio видає візуальні підказки про помилки чи потенційні проблеми коду ще в процесі його написання. Розробник може одразу звернути увагу на наявність помилки та навіть прочитати відомості про неї і можливі шляхи вирішення

у спливаючій підказці. Таким чином, середовище розробки допомагає скоротити кількість помилок коду ще на етапі його написання.

Для пришвидшення написання коду розробнику на допомогу приходить вбудований набір функцій IntelliSense. Його дія полягає в тому, що при написанні коду середовище автоматично видає список можливих продовжень стрічки. Так розробник може швидко знайти потрібну функцію, а програма сама додасть обраний відрізок коду.

Для підвищення адаптивності додатків, розробникам доступні базові функції рефакторингу: інтелектуальне перейменування змінних, об’єднання стрічок коду у метод, зміна параметрів чи порядку методів та інші. Вони є зручними за необхідності деякої стандартизації коду, задля полегшення розуміння коду іншими розробниками, якщо, як і у випадку розроблюваної системи захисту, планується подальша інтеграція підсистеми у загальну програму.

Не менш важливою перевагою Visual Studio, з точки зору вибору її як засобу розробки для розроблюваної системи захисту є те, що вона є безкоштовною. У середовища є декілька версій, деякі з яких потребують оплати підписки, однак Community Edition попри те, що є безкоштовним, має увесь необхідний для створення розроблюваної системи захисту набір бібліотек, функцій та навіть можливість завантаження додаткових плагінів [39].

Таким чином, використання Visual Studio у якості середовища розробки дозволить скоротити кількість помилок коду, пришвидшити процес написання програми та покращити її адаптивність.

3.2 Проектування та розробка алгоритму та модулів підсистеми захисту

Роботу над програмною реалізацією підсистеми було розпочато із проектування алгоритму роботи програми, на основі структурної схеми роботи розробленого удосконаленого алгоритму, що зображено на рисунку Б1 у додатку Б.

Блок-схему алгоритму роботи програмної підсистеми для тестування роботи удосконаленого алгоритму обчислення редакційної відстані Дамероу-Левенштейна представлено на рисунку Б2 у додатку Б.

Далі наведено детальний опис роботи алгоритму:

Крок 1. Завантаження порівнюваних кодових послідовностей.

Крок 2. Виконання порівняння послідовностей за алгоритмом Вагнера-Фішера для обчислення відстані Левенштейна.

Крок 3. Вивід результату порівняння – відстані Левенштейна між порівнюваними послідовностями.

Крок 4. Вивід часу витраченого на обробку вхідних даних алгоритмом.

Далі для виконання перевірки за оригінальним алгоритмом Вагнера-Фішера для відстані Дамероу-Левенштейна:

Крок 5. Виконання порівняння послідовностей за алгоритмом Вагнера-Фішера для обчислення відстані Дамероу-Левенштейна.

Крок 6. Вивід результату порівняння – відстані Дамероу-Левенштейна між порівнюваними послідовностями.

Крок 7. Вивід часу витраченого на обробку вхідних даних алгоритмом.

Крок 8. Виконання порівняння результатів обчислення редакційної відстані між оригінальним алгоритмом для відстані Левенштейна та оригінальним алгоритмом для відстані Дамероу-Левенштейна.

Крок 9. Виконання порівняння часу необхідного для проведення порівняння між оригінальним алгоритмом для відстані Левенштейна та оригінальним алгоритмом для відстані Дамероу-Левенштейна.

Крок 10. Вивід відсоткового відношення приросту точності обчислення редакційної відстані.

Крок 11. Вивід відсоткового відношення приросту часової ефективності .

Для виконання перевірки за удосконаленим алгоритмом Вагнера-Фішера для відстані Дамероу-Левенштейна (коли «Очищення коду» вибрано):

Крок 12. Приведення усіх буквених символів обох кодових послідовностей до верхнього регістру.

Крок 13. Очищення кодових послідовностей від невід’ємних виразів коду, шляхом заміни відповідних визначених наборів символів на null.

Крок 14. Вивід очищених версій оригінального коду та коду копії.

Крок 15. Виконання порівняння послідовностей за алгоритмом Вагнера-Фішера для обчислення відстані Дамероу-Левенштейна.

Крок 16. Вивід результату порівняння – відстані Дамероу-Левенштейна між порівнюваними послідовностями.

Крок 17. Вивід часу витраченого на обробку вхідних даних алгоритмом.

Крок 18. Виконання порівняння результатів обчислення редакційної відстані між оригінальним алгоритмом для відстані Левенштейна та удосконаленим алгоритмом для відстані Дамероу-Левенштейна.

Крок 19. Виконання порівняння часу необхідного для проведення порівняння між оригінальним алгоритмом для відстані Левенштейна та удосконаленим алгоритмом для відстані Дамероу-Левенштейна.

Крок 20. Вивід відсоткового відношення приросту точності обчислення редакційної відстані.

Крок 21. Вивід відсоткового відношення приросту часової ефективності.

Далі розглянемо програмну реалізацію основного модуля підсистеми.

Умовою для початку роботи із програмою є завантаження файлів оригіналу коду та його копії у відповідні поля (рисунок 3.1)

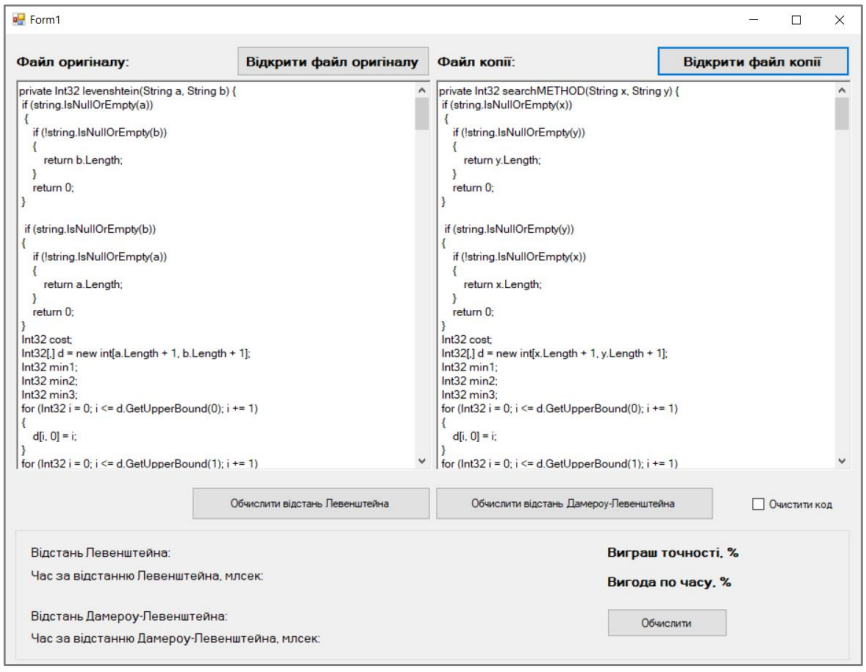


Рисунок 3.1 – Вигляд головного вікна програми реалізації підсистеми порівняння файлів вихідного коду

Найбільший інтерес собою представляють методи реалізації алгоритму обчислення відстані Левенштейна levenstain() та відстані Дамероу-Левенштейна damerou-levenstain().

Для того щоб почати перевірку файлів вихідного коду за алгоритмом для визначення відстані Левенштейна потрібно натиснути кнопку «Обчислити відстань Левенштейна», яка автоматично викликає метод levenstain():

```
void levenstain()
{
  string a = richTextBox1.Text;
   string b = richTextBox2.Text;
   Int32 dist = 0;
   timer1Start = TimeUtils.GetMicroseconds();
if (string.IsNullOrEmpty(a))
{
   if (!string.IsNullOrEmpty(b))
   {
   dist = b.Length;
   }
   dist = 0;
 }
if (string.IsNullOrEmpty(b))
{
  if (!string.IsNullOrEmpty(a))
  {
    dist = a.Length;
  }
  dist = 0;
  }
  Int32 cost;
  Int32[,] d = new Int32[a.Length + 1, b.Length + 1];
  Int32 min1;
  Int32 min2;
  Int32 min3;
  for (Int32 i = 0; i <= d.GetUpperBound(0); i += 1)
  {
    d[i, 0] = i;
   }
  for (Int32 i = 0; i <= d.GetUpperBound(1); i += 1)
   {
     d[0, i] = i;
   }
   for (Int32 i = 1; i <= d.GetUpperBound(0); i += 1)
   {
     for (Int32 j = 1; j <= d.GetUpperBound(1); j += 1)
      {
        cost = Convert.ToInt32(!(a[i - 1] == b[j - 1]));
```

```
                min1 = d[i - 1, j] + 1;
                min2 = d[i, j - 1] + 1;
                min3 = d[i - 1, j - 1] + cost;
                d[i, j] = Math.Min(Math.Min(min1, min2), min3);
            }
        }
    dist = d[d.GetUpperBound(0), d.GetUpperBound(1)];
    timer1End = TimeUtils.GetMicroseconds();
    label5.Text = dist.ToString();
    label8.Text = (timer1End - timer1Start).ToString();
    MessageBox.Show("Відстань Левенштейна: " + dist + ". Часу витрачено (млсек): " + (timer1End
- timer1Start).ToString());
    }
```

Спершу метод зчитує код оригіналу та копії як дві стрічки. Далі запускається таймер для підрахунку часу, що потрібен буде алгоритму для обчислення редакційної відстані. Далі слідує перевірка стрічок за рядом умов:

1. Якщо перша стрічка a пуста, а друга стрічка b – не пуста, то відстань Левенштейна буде рівна довжині стрічки b.

2. Якщо перша стрічка b пуста, а друга стрічка a – не пуста, то відстань Левенштейна буде рівна довжині стрічки a.

3. Якщо і стрічка a, і стрічка b пусті, то відстань Левенштейна буде рівною 0.

4. Якщо і стрічка a, і стрічка b не пусті то відстань Левенштейна обчислюється за алгоритмом Вагнера-Фішера

Після закінчення обчислення редакційної відстані, метод фіксує час закінчення порівняння. Після цього метод виводить результат роботи: обчислену відстань Левенштейна та час, затрачений на обчислення (рисунок 3.2).

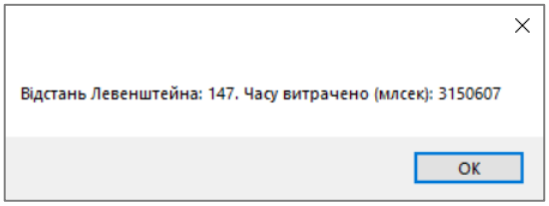


Рисунок 3.2 – Вигляд вікна виводу результатів порівняння кодових послідовностей

Для того, щоб провести обчислення відстані Дамероу-Левенштейна потрібно натиснути кнопку «Обчислити відстань Дамероу-Левенштейна», яка викликає метод damerou-levenstain():

```
void damerou_levenstain()
{
string m = "";
string n = "";
int[] current;
int[] previous;
int[] transposition;
int max = 1024;
int dist = 0;
if (richTextBox1.Text.Length > richTextBox2.Text.Length)
 {
  m = richTextBox1.Text.ToUpper
  n = richTextBox2.Text.ToUpper();
 }
else
 {
  m = richTextBox2.Text.ToUpper();
  n = richTextBox1.Text.ToUpper();
 }
timer2Start = TimeUtils.GetMicroseconds();
if (checkBox1.Checked)
{
 m = proceed(m);
 n = proceed(n);
}
int m_length = m.Length;
int n_length = n.Length;
int max_length;
if (m_length > n_length) max_length = m_length;
else max_length = n_length;
max = max_length;
current = new int[max_length + 1];
previous = new int[max_length + 1];
transposition = new int[max_length + 1];
if (m_length == 0)
dist = n_length;
if (n_length == 0)
dist = m_length;
if (m_length > n_length)
{
 string tmp = m;
 m = n;
 n = tmp;
 m_length = n_length;
 n_length = n.Length;
}
if (max < 0) max = n_length;
if (n_length - m_length > max)
dist = max + 1;
if (m_length > current.Length)
```

```
{
 current = new int[m_length + 1];
 previous = new int[m_length + 1];
 transposition = new int[m_length + 1];
}
for (int i = 0; i <= m_length; i++)
previous[i] = i;
char last_n_char = (char)0;
for (int i = 1; i <= n_length; i++)
{
 char n_char = n[i - 1];
 current[0] = i;

 int from = Math.Max(i - max - 1, 1);
 int to = Math.Min(i + max + 1, m_length);
 char last_m_char = (char)0;
 for (int j = from; j <= to; j++)
 {
  char m_char = m[j - 1];
  int cost = m_char == n_char ? 0 : 1;
  int value = Math.Min(Math.Min(current[j - 1] + 1, previous[j] + 1), previous[j - 1] + cost);
  if (m_char == last_n_char && n_char == last_m_char)
  value = Math.Min(value, transposition[j - 2] + cost);
  current[j] = value;
  last_m_char = m_char;
 }
 last_n_char = n_char;
 int[] temporary = transposition;
 transposition = previous;
 previous = current;
 current = temporary;
}
dist = previous[m_length];
timer2End = TimeUtils.GetMicroseconds(); // фіксація часу закінчення роботи алгоритму

MessageBox.Show("Відстань Дамероу-Левенштейна: " + dist + ". Часу витрачено (млсек): " +
(timer2End - timer2Start).ToString());
label6.Text = dist.ToString();
label7.Text = (timer2End - timer2Start).ToString();
}
```

Першою дією метод обирає між стрічками ту, що є меншою за довжиною – вона і буде задавати довжину виконання циклу при порівнянні стрічок. Після цього усі символи обох порівнюваних стрічок приводяться до верхнього регістру. Перед початком порівняння метод запускає таймер для фіксації часу обробки даних. Далі слідує перевірка стрічок згідно наступних умов:

1. Якщо m_length = 0, а n_length ≠ 0, то відстань Дамероу-Левенштейна буде рівна довжині стрічки n.

2. Якщо n_length = 0, а m_length ≠ 0, то відстань Дамероу-Левенштейна буде рівна довжині стрічки m.

3. Якщо і m_length = 0, і n_length = 0, то відстань Дамероу-Левенштейна буде рівною 0.

4. Якщо n_length ≠ 0, і m_length ≠ 0 то відстань Дамероу-Левенштейна обчислюється за алгоритмом Вагнера-Фішера з урахуванням вартості транспозиції, якщо її застосування було виявлено.

По закінченню обчислень, алгоритм зупиняє таймер та виводить результати операції порівняння: обчислену відстань Дамероу-Левенштейна та час затрачений на обробку даних алгоритмом.

У випадку якщо чекбокс «Очистити код» вибрано, то відстань Дамероу-Левенштейна обчислюється згідно удосконаленого алгоритму, що передбачає виконання методу proceed() для обох порівнюваних стрічок коду перед початком їх порівняння за алгоритмом Вагнера-Фішера:

```
private string proceed(string inData)
{
 inData = inData.Replace("INT32", "");
 inData = inData.Replace("INT", "");
 inData = inData.Replace(" ", "");
 inData = inData.Replace("\n", "");
 inData = inData.Replace("SYSTEM", "");
 inData = inData.Replace("STRING", "");
 inData = inData.Replace("VAR", "");
 inData = inData.Replace("WINDOWS", "");
 inData = inData.Replace("DOUBLE", "");
 inData = inData.Replace(".", "");
 inData = inData.Replace("(", "");
 inData = inData.Replace(")", "");
…
 return inData;
}
```

Даний метод покликаний видалити з коду усі вирази та символи, що завчасно є однаковими у обох порівнюваних послідовностях, оскільки є невід’ємними від коду для тої мови програмування, якою написано код, що проходить перевірку (назви

змінних, модифікатори доступу тощо), а також пробіли, символи закриття та відкриття циклів («{» та «}») та операцій («(» та «)»), «;» тощо. Всі вказані символи та послідовності символів метод заміняє на null. Таким чином, у цикл для порівняння стрічок потрапляють не оригінальні послідовності, а позбавлені усіх завчасно однакових елементів, порівнянням яких можна знехтувати, та відповідно значно коротші за оригінальні. Це відповідно має позитивно пливати на швидкість обчислення відстані Дамероу-Левенштейна.

Після того, як було виконано обчислення за обома алгоритмами, при натисненні кнопки «Обчислити» викликається метод count():

```
void count()
    {
      int origOps = Convert.ToInt32(label8.Text);
      int modOps = Convert.ToInt32(label7.Text);
      float varOps = 100 - (float)100 * ((float)modOps / (float)origOps);
      label11.Text = varOps.ToString();
      int origDst = Convert.ToInt32(label5.Text
      int modDst = Convert.ToInt32(label6.Text);
      float varDst = 100 - (float)100 * ((float)modDst / (float)origDst);
      label12.Text = varDst.ToString();
    }
```

У даному методі здійснюється порівняння результатів обчислення редакційної відстані за алгоритмом для відстані Левенштейна та відстані Дамероу-Левенштейна шляхом знаходження відсоткового відношення покращення точності обчислення відмінності між порівнюваними файлами. Аналогічно здійснюється обрахунок відсоткового відношення скорочення часу обчислення редакційної відстані за алгоритмом для відстані Дамероу-Левенштейна відносно алгоритму для відстані Левенштейна (рисунок 3.3).

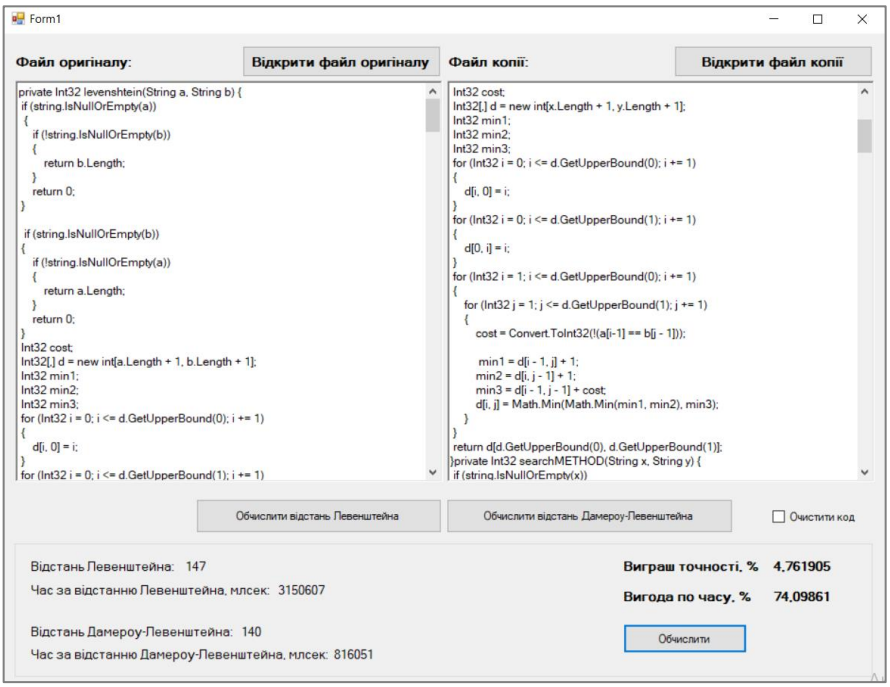


Рисунок 3.3 – Вигляд головного вікна програми реалізації підсистеми порівняння файлів вихідного коду при виконанні порівняння результатів отриманих за різними алгоритмами

Таким чином, розроблена підсистема дозволяє провести порівняння кодових послідовностей за оригінальними алгоритмами для відстані Левенштейна та Дамероу-Левенштейна, а також за удосконаленим алгоритмом для відстані Дамероу-Левенштейна, а також провести порівняння результатів цих обчислень за показниками точності та швидкості виконання перевірки.

3.3 Тестування роботи розробленого рішення в рамках створеної підсистеми захисту

З метою визначити чи було досягнуто бажаного результату завдяки реалізації розробленого удосконаленого рішення для алгоритму обчислення відстані Дамероу-Левенштейна, було виконано тестування на 10 парах файлів вихідного коду програм різної довжини за кількістю символів із пробілами. Кожну пару послідовностей становили оригінальний файл та копія цього файлу, що містила код оригіналу після застосування до нього простих процедур рефакторингу. Ступінь рефакторингу застосований до файлу копії у кожній із трестованих пар також був різним.

Для проведення аналізу та виведення висновків про результати тестування було використано дані обчислень виграшу по часу та точності порівняння, обчислені розробленою підсистемою захисту (див. рис. 3.4).

В першу чергу було проведено аналіз щодо досягнутих показників точності порівняння. Результати, отримані під час тестування наведено у таблиці 3.1.

Таблиця 3.1 – Результати тестування удосконаленого рішення для алгоритму порівняння файлів вихідного коду за показниками точності порівняння кодових послідовностей

Номер тесту	Довжина коду	Редакційна відстань			Відношення	
		D_L	D_{DL}	D_{DL_m}		
1	859	40	38	38	D_{DL}/D_L	5
	861				D_{DL_m}/D_L	5
2	2167	82	77	77	D_{DL}/D_L	6,097560976
	2171				D_{DL_m}/D_L	6,097560976
3	3475	126	121	120	D_{DL}/D_L	3,968253968
	3481				D_{DL_m}/D_L	4,761904762
4	4129	145	140	140	D_{DL}/D_L	3,448275862
	4136				D_{DL_m}/D_L	3,448275862
5	4578	147	140	140	D_{DL}/D_L	4,761904762
	4585				D_{DL_m}/D_L	4,761904762
6	5689	159	155	155	D_{DL}/D_L	2,51572327
	5693				D_{DL_m}/D_L	2,51572327
7	6589	168	138	138	D_{DL}/D_L	17,85714286
	6580				D_{DL_m}/D_L	17,85714286
8	7825	172	155	155	D_{DL}/D_L	9,88372093
	7832				D_{DL_m}/D_L	9,88372093

Продовження таблиці 3.1

9	8992	188	187	187	D_{DL}/D_L	0,531914894
	8995				D_{DL_m}/D_L	0,531914894
10	10001	199	188	188	D_{DL}/D_L	5,527638191
	9989				D_{DL_m}/D_L	5,527638191

Згідно отриманих результатів тестування можна зробити наступні висновки:

1. У всіх тестах вдалося досягнути більшої точності порівняння послідовностей по відношенню до оригінального алгоритму для відстані Левенштейна.

2. Не виявлено закономірностей щодо виграшу у точності порівняння. Це пояснюється тим, що на даний показник впливають одразу 2 фактори: довжина порівнюваних послідовностей а кількістю символів та степінь зміни файлу копії відносно оригіналу шляхом рефакторингу.

3. Похибка обчислень редакційної відстані при порівнянні за удосконаленим алгоритмом обчислено за формулою:

$$\Delta = \frac{\sum_n^1 |R_{n\,(mod)} - R_{n\,(org)}|}{n} * 100\% \quad (3.1)$$

де n – номер тесту;

$R_{n\,(mod)}$- відношення точності порівняння для удосконаленого алгоритму для n-го тесту;

$R_{n\,(org)}$- відношення точності порівняння для оригінального алгоритму для n-го тесту.

Результат оцінки похибки обчислень згідно отриманих результатів тестування становить $\Delta \approx 7{,}94\%$.

Далі було виконано аналіз показників швидкості обробки вхідних даних удосконаленим алгоритмом, у порівнянні із оригінальними алгоритмами для визначення відстані Левенштейна та Дамероу-Левенштейна. Результати відповідних тестувань наведено у таблиці 3.2.

Таблиця 3.2 – Результати тестування удосконаленого рішення для алгоритму порівняння файлів вихідного коду за показниками швидкості обробки алгоритму при порівнянні кодових послідовностей

Номер тесту	Довжина коду	Час виконання обчислень, млсек			Відношення	
		T_L	T_{DL}	T_{DL_m}		
1	859	102925	28948	8822	T_{DL}/T_L	71,87467
	861				T_{DL_m}/T_L	91,42871
2	2167	633668	182305	52675	T_{DL}/T_L	71,2302
	2171				T_{DL_m}/T_L	93,2654
3	3475	1682056	466939	120192	T_{DL}/T_L	57,84353
	3481				T_{DL_m}/T_L	93,66698
4	4129	2334217	659690	171281	T_{DL}/T_L	62,65443
	4136				T_{DL_m}/T_L	91,58
5	4578	2849139	811315	208246	T_{DL}/T_L	61,81786
	4585				T_{DL_m}/T_L	91,0591
6	5689	3528154	1256846	536981	T_{DL}/T_L	61,06611
	5693				T_{DL_m}/T_L	84,91457
7	6589	4893269	1832684	682462	T_{DL}/T_L	57,66992
	6580				T_{DL_m}/T_L	85,77025
8	7825	5569852	2346975	765895	T_{DL}/T_L	54,60267
	7832				T_{DL_m}/T_L	85,95908
9	8992	6489325	3256891	846893	T_{DL}/T_L	49,81156
	8995				T_{DL_m}/T_L	86,94944
10	10001	7569862	4362589	925164	T_{DL}/T_L	42,36898
	9989	102925			T_{DL_m}/T_L	86,4573

У всіх тестах вдалося досягнути кращих показників швидкості обробки вхідних даних при порівнянні послідовностей одночасно по відношенню до оригінальних алгоритму для відстані Левенштейна та відстані Дамероу-Левенштейна.

Згідно отриманих даних тестування було побудовано графік прогресії витрат часу на обробку кожним із досліджуваних алгоритмів вхідних даних залежно від збільшення довжини цих даних за кількістю символів із пробілами. Відповідний графік наведено на рисунку 3.4.

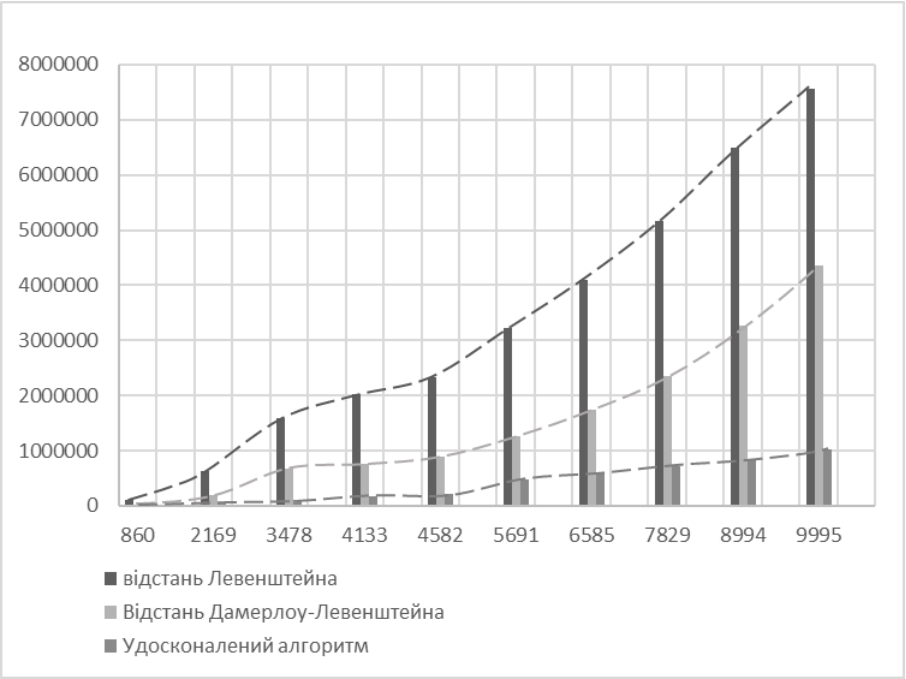


Рисунок 3.4 – Графік прогресії часу обробки вхідних даних досліджуваними алгоритмами

З рисунку 3.5 видно що дані графіки не мають ідеальної параболічної форми. Це може бути пов’язано із похибкою, яку вносить продуктивність процесора. Якщо провести багаторазову перевірку роботи кожним із алгоритмів для однакової пари кодів, то показники часу обробки алгоритмами вхідних даних будуть різними з невеликим відхиленням. Результати такої перевірки, для пари кодів із тесту №5, зокрема, наведено у таблиці 3.3

Таблиця 3.3 – Результати обчислення часу виконання алгоритму для пари тестових файлів №5

Номер перевірки	Час роботи алгоритму, млсек		
	T_L	T_{DL}	T_{DL_m}
1	3252893	957719	211890
2	3252893	845418	210642
3	2829042	813122	211326
4	2997969	830854	220368
5	2721460	830854	214407
6	2747172	1119725	211041
7	2723149	811416	211120
8	2743802	822704	210684
9	2718494	843638	212124
10	2757746	820375	211285

Попри це, якщо порівняти графіки на рисунках 3.5 та 2.5 то можна простежити, що все ж, як і прогнозувалося, завдяки розробленому удосконаленню до алгоритму визначення відстані Дамероу-Левенштейна, вдалося досягнути сповільнення збільшення часу обробки вхідних даних при збільшенні об’єму цих вхідних даних, а отже значно покращити часову ефективність алгоритму порівняння текстів файлів вихідного коду програм. Якщо співвіднести між собою результати порівняння часу затраченого на роботу алгоритму між алгоритмом для відстані Левенштейна та відстані Дамероу-Левенштейна і відстані Левенштейна та удосконаленого алгоритму, то середній приріст ефективності між оригінальним та удосконаленим алгоритмами для відстані Дамероу-Левенштейна складає 30,01%.

Таким чином, завдяки розробленому удосконаленню до алгоритму оцінки відстані Дамероу-Левенштейна вдалося досягнути покращення показників точності порівняння файлів вихідного коду, що мало прояв для усіх тестових зразків пар файлів вихідного коду. Похибка обчислень становить близько 8%. Це нівелюється досягнутими показниками покращення часової ефективності порівняння за рахунок

застосування розробленого рішення – приріст ефективності по відношенню до оригінального алгоритму сягнув близько 30%.

3.4 Висновки до розділу 3

У даному розділі було виконано програмну реалізацію підсистеми виявлення автентичності файлів вихідного коду на основі розробленого удосконаленого методу. Спершу було здійснено вибір засобів розробки. Основною мовою програмування для підсистеми захисту було обрано C#. Розробку модулів програмної підсистеми було виконано у середовищі розробки Visual Studio із використанням технології Windows Forms.

Далі було виконано розробку основного модуля підсистеми. Реалізовано методи для обчислення відстані Левенштейна за алгоритмом Вагнера-Фішера, відстані Дамероу-Левенштейна за оригінальним алгоритмом Вагнера-Фішера та із запропонованим рішенням для удосконалення методу порівняння, відповідно до особливостей предметної області. Відповідно підсистема дозволяє обрахувати результати порівняння двох версій файлу вихідного коду за трьома методами порівняння та обчислює значення приросту за визначеними показниками успішності реалізації рішення: точність порівняння та швидкість обробки вхідних даних алгоритмом.

Після цього було виконано тестування розробленого рішення засобами створеної підсистеми. За показниками точності порівняння вдалося у більшості протестованих зразків отримати результати співставні із результатами обчислень за оригінальним алгоритмом Дамероу-Левенштейна. Похибка порівняння становить близько 8%. За часовою ефективністю також вдалося досягнути покращення за показником швидкості обробки вхідних даних алгоритмом – приріст ефективності становить близько 30%. Таким чином, за рахунок значного приросту швидкості обробки алгоритму незначна похибка порівняння нівелюється.

4. ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

При проведенні оцінки економічного потенціалу розробки основний фокус розподіляється між виявленням доцільності втілення нових ідей технологій та їх здійсненністю в промисловому масштабі. В рамках оцінювання обов’язково розглядаються наступні блоки питань: переваги для споживачів; характеристики можливого ринку; основні конкуренти; здійсненність ідеї; забезпечення ресурсами.

Успішність продукту визначає його конкурентоспроможність. Тому надійний захист інтелектуальної власності, покладеної в основу розробки, є важливим фактором зменшення ризику передчасного згасання циклу продажів нового продукту.

Особливо корисні при проведенні порівняльного аналізу технологій та їх ранжуванні за комерційним потенціалом або відповідним ризиками, не лише якісні, а й кількісні методи оцінки комерційного потенціалу технологій. Критерії, що використовуються для оцінки комерційного потенціалу системи наведено в таблиці 4.1. При використанні такого підходу за кожною із ознак виставляється певний максимальний бал і ставляться конкретні оцінки для проекту що оцінюється. Після виявлення всіх ознак можуть вводитися коефіцієнти "вагомості" даної ознаки або всієї групи факторів (наприклад, що характеризують рівень технологічних переваг) в загальному комплексі розглянутих параметрів. Для підвищення точності оцінювання комерційного потенціалу, дана процедура може виконуватися паралельно кількома експертами в даній галузі, після чого їх бали додаються та знаходиться середньоарифметична оцінка комерційного потенціалу.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки інформаційної системи та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Бали	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів не має
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресур- си, які відсутні. Джерела фін. ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фін. відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовують- ся у військ. пром. комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використ. у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років.	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років.
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Отримані при оцінці комерційного потенціалу результати відображено у таблиці 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки інформаційної системи

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами:		
1	3	2	3
2	3	4	4
3	2	3	3
4	3	3	3
5	3	2	2
6	1	1	2
7	3	4	3
8	3	4	2
9	2	3	2
10	4	4	3
11	2	3	4
12	1	1	2
Сума балів	$СБ_1$=30	$СБ_2$=34	$СБ_3$=33
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = 32$		

Таблиця 4.3 – Рівні комерційного потенціалу розробки інформаційної системи

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Відповідно до таблиці 4.3 було зроблено висновок щодо рівня комерційного потенціалу розробки: оскільки середньоарифметична сума балів отриманих на фазі оцінки комерційного потенціалу становить 32 бали, що відповідає рівню комерційного потенціалу – вище середнього.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідницько-конструкторської) та конструкторсько-технологічної роботи

Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи складається з наступних етапів:

1. Розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи.

2. Розрахунок загальних витрат на виконання даної роботи.

3. Прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

4. Розрахунок витрат, які безпосередньо стосуються виконавців даного розділу НДДКР

Отже, першим етапом розраховується основна заробітна плата розробників за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \qquad (4.1)$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

Тр – число робочих днів в місяці, 22 днів;

t – число днів роботи розробника (дослідника).

Над створенням підсистеми працював один розробник. Результати обчислення заробітної плати для нього наведено у таблиці 4.4.

Таблиця 4.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число робочих днів	Витрати на заробітну плату, грн.
Розробник	30 000,00	1 363,63	15	20 454,54
Всього				20 454,54

Витрати на основну заробітну плату робітників ($З_р$) розраховуються на основі норм часу, які необхідні для виконання даної роботи:

$$З_р = \sum_{1}^{n} t_i * С_i * K_c \quad (4.2)$$

де $t_ш$ – норма часу (трудомісткість) на виконання конкретної роботи, годин;

n – число робіт по видах та розрядах;

$К_c$ – коефіцієнт співвідношень, який установлений в даний час Генеральною тарифною угодою між Урядом України і профспілками;

$С_i$ – погодинна тарифна ставка робітника відповідного розряду, який виконує відповідну роботу, грн./год.

$С_i$ визначається за формулою:

$$С_i = \frac{М_н * К_i}{Т_р * Т_{зм}} \quad (4.3)$$

де $М_н$ – мінімальна місячна оплата праці, грн., Мн = 6700 грн (грудень 2022 року);

$К_i$ – тарифний коефіцієнт робітника відповідного розряду;

$Т_р$ – число робочих днів в місяці, $Т_р$ = 22 дні;

$Т_{зм}$ – тривалість зміни, $Т_{зм}$ = 8 годин.

Погодинна тарифна ставка відповідно до формули 4.3 становить 39,128 грн/год.

Розраховані витрати на заробітну плату робітників відображено в табл. 4.5.

Таблиця 4.5 – Витрати на основну заробітну плату робітників

Найменування робіт	Трудомісткість, год.	Розряд роботи	Погодинна тарифна ставка, грн.	Величина оплати на робітника, грн
Розробка веб-додатку	120	6	39,128	8 451,648
Всього				8 451,648

Додаткова заробітна плату будемо розраховувати як 12 % від основної заробітної плати розробників та робітників відповідно до формули:

$$З_Д = \frac{З_О \cdot 12\%}{100\%} \quad (4.4)$$

Згідно формули 4.4, $З_Д = 20\,454{,}54 \cdot 0{,}12 = 2\,454{,}54$ (грн).

Згідно діючого законодавства, нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_{ЗП} = \frac{(З_О + З_Р + З_Д) \cdot 22\%}{100\%} \quad (4.5)$$

Підставивши отримані значення в попередніх крокам отримаємо наступну суму:

$Н_{зп} = (20\,454{,}54 + 8\,451{,}648 + 2\,454{,}54) * 0.22 = 31\,360{,}7 * 0.22 =$
$6\,899{,}36$ (грн).

Амортизація обладнання, що використовувалось для розробки розраховується за формулою:

$$А = \frac{Ц * На}{100} \cdot \frac{T}{12} \quad (4.6)$$

де Ц – балансова вартість обладнання, грн;

Т – термін корисного використання обладнання згідно податкового законодавства, років;

$Н_а$ – норма амортизації, приймемо за 20 %.

Перелік використаних ресурсів, обладнання, їх балансова вартість та відповідні значення амортизаційних відрахувань відображено в таблиці 4.6.

Таблиця 4.6 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів

Найменування обладнання	Балансова вартість, грн.	Норма амортизації %	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Комп'ютер	20000	20	1	333,3
Приміщення	160000	20	1	2 666,666
Всього				2 999,99

Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В * П * Ф * К_п \quad (4.7)$$

де В — вартість 1 кВт-години електроенергії, В = 492,791 грн./кВт (у грудні 2022 року);

П — встановлена потужність обладнання, кВт. П = 0.4 кВт;

Ф — фактична кількість годин роботи обладнання, годин – .

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,6.

$В_е = 492,791 * 0.4 * 120 * 0.6 = 14\,192,38$ (грн)

Інші витрати охоплюють: загально виробничі витрати, адміністративні витрати, витрати на збут тощо. Інші витрати доцільно приймати як 100...300% від суми основної заробітної плати розробників та робітників. В даній роботі величина інших витрат складе:

$В_{ін}$ = 20 454,54 * 2 + 8 451,648 * 2 = 57 812,376 (грн)

Сума всіх попередніх статей витрат дає загальні витрати на проведення розробки:

$В_{заг}$ = 20 454,54 + 8 451,648 + 2 454,54 + 6 899,36 + 2 999,99 + 14 192,38 + 57 812,376 = 113 264,83 (грн).

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної МКР здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (4.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної МКР $\beta = 0{,}1...0{,}9$. Приймемо β = 0,5, так як розробка, на даний момент часу, знаходиться на стадії дослідного зразка. Отже, значення загальних витрат буде дорівнювати:

$ЗВ = \frac{113\ 264{,}83}{0{,}5} = 226\ 529{,}67$ (грн).

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Проведено кількісне прогнозування вигоди, що можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

Збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right), \quad (4.9)$$

де $\Delta Ц_0$ — покращення основного оціночного показника від впровадження результатів розробки у даному році.

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

p — коефіцієнт, який враховує рентабельність продукту, p = 0,25;

υ — ставка податку на прибуток. У 2020 році — 18%.

В результаті впровадження наукової розробки полегшується проведення розслідувань порушення автентичності файлів вихідного коду із покращенням точності порівняння, що дозволяє підвищити ціну його реалізації на 1000 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 200 шт., протягом другого року – ще на 300 шт., протягом третього року – ще на 400 шт.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 0 шт., а її ціна – 3500 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом першого року складе:

$$\Delta П_1 = [1000 \cdot 0 + (3500 + 1000) \cdot 200] \cdot 0.8333 \cdot 0.25 \cdot (1 - 0.18) =$$

153 743,85 (грн)

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом другого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_2 = [1000 \cdot 0 + (3500 + 1000) \cdot (200 + 300)] \cdot 0.8333 \cdot 0.25 \cdot (1 - 0.18) = 384\,359{,}625 \text{ (грн)}$$

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом третього року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_3 = [1000 \cdot 0 + (3500 + 1000) \cdot (200 + 300 + 400)] \cdot 0.8333 \cdot 0.25 \cdot (1 - 0.18) = 691\,847{,}325 \text{ (грн)}$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахований у попередньому підрозділі комерційний ефект від можливого впровадження розробки, ще не означає, що ця розробка реально буде впроваджена. Якщо збільшення прогнозованого прибутку від впровадження результатів наукової розробки є вигідним для підприємства (організації), то це ще не означає, що інвестор погодиться фінансувати дану розробку. Інвестор погодиться вкладати кошти у реалізацію даної наукової розробки тільки за певних умов.

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1. Розрахунок теперішньої вартість інвестицій PV, що вкладаються в наукову розробку. Такою вартістю, можна вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, розраховану нами раніше за формулою (4.8), тобто будемо вважати, що ЗВ = PV=226 529,67 грн.

2. Розрахунок очікуваного збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження.

3. Розрахунок абсолютної ефективності вкладених інвестицій $Е_{абс}$.

$$E_{abc} = (ПП - PV) \qquad (4.10)$$

де *ПП* – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.; *PV* – теперішня вартість інвестицій, *PV = ЗВ*.

У свою чергу, приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^{\tau}} \quad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої МКР, грн;

Т – період часу, протягом якою виявляються результати впровадженої МКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках).

($ПП = \frac{153\,743{,}85}{(1+0{,}1)^3} + \frac{384\,359{,}625}{(1+0{,}1)^4} + \frac{691\,847{,}325}{(1+0{,}1)^5} = \frac{153\,743{,}85}{1{,}331} + \frac{384\,359{,}625}{1{,}4641} + \frac{691\,847{,}325}{1{,}61051} =$

807 615,582 (грн).

Тоді

$$Е_{абс} = 807\,615{,}582 - 226\,529{,}67 = 581\,085{,}86 \text{ (грн)}.$$

Оскільки $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток.

4. Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього використаємо формулу:

$$E_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \quad (4.12)$$

де $Т_Ж$ – життєвий цикл наукової розробки, роки.

$$E_{\text{в}} = \sqrt[3]{1 + \frac{581\,085{,}86}{226\,529{,}67}} - 1 = 0{,}44 \qquad (4.13)$$

Розрахуємо мінімальну (бар'єрну) ставку дисконтування:

$$\tau = d + f \qquad (4.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках (в 2022 році в Україні d = 0.095);

f-показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0.07).

Отже, $\tau = 0.095 + 0.07 = 0.102$

Оскільки $E_{\text{в}} > \tau$, тобто 44% >10,2%, інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

5. Розраховуємо термін окупності вкладених в реалізацію наукового проекту інвестицій:

$$T_{ok} = \frac{1}{E_{\text{в}}} \qquad (4.16)$$

Відповідно до формули наведеної вище термін окупності буде такий: $T_{ok} = \frac{1}{0{,}44} = 2{,}27$

Оскільки, $T_{ок}$<3..5-ти років, то фінансування даної наукової розробки є доцільним.

4.5 Висновки до розділу 4

У даному розділі було здійснено оцінку комерційного потенціалу розробки. Визначено, що він є вище середнього, що свідчить про велику ймовірність успішного комерційного впровадження системи на ринок та відповідно можливості отримання прибутку від її використання.

Також, розраховано витрати на виконання наукової роботи та впровадження її результатів, а саме: основна заробітна плата кожного з розробників, сума яких рівна 20 454,54 грн та робітників – 8 451,648грн; додаткова заробітна плата – 2 454,54 грн; нарахування на заробітну плату розробників та робітників – 6 899,36 грн; амортизація обладнання – 2 999,99 грн; витрати на силову електроенергію – 14 192,38 грн та інші витрати – 57 812,376 грн.

Обчислено загальні витрати на розробку, що складають 113 264,83 грн. Оскільки розробка знаходиться на стадії розробки дослідного зразка, то прогнозовані загальні витрати становитимуть 226 529,67 грн.

Визначено комерційні ефекти від реалізації результатів розробки. Для виконання даної наукової роботи та впровадження її результатів необхідно затратити 1 рік, а основні позитивні результати від впровадження розробки очікуються протягом 3-ох років після її впровадження.

Також розраховано збільшення чистого прибутку для кожного року, протягом яких очікується отримання позитивних результатів, для 1-го року – 153 743,85 грн; для 2-го – 384 359,625 грн; для 3-го – 691 847,325 грн.

Спрогнозовано ефективність вкладених інвестицій та період їх окупності. Оскільки, абсолютна ефективність $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток. Розраховано відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$ та мінімальну (бар'єрну) ставку дисконтування. Оскільки $E_в > \tau$, тобто 44% >10,2%, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Також розраховано термін окупності вкладених в реалізацію наукового проекту інвестицій який складає 2,27 роки. Оскільки $T_{ок}$<3..5-ти років, то фінансування даної наукової розробки є доцільним.

ВИСНОВКИ

У даній роботі було виконано розробку рішення для удосконалення методу виявлення порушення автентичності у файлах вихідного коду програм. Було визначено, що у зв’язку із недосконалістю існуючого міжнародного законодавства щодо захисту авторських прав на об’єкти інтелектуальної власності у сфері розробки програмного забезпечення, а саме розглядання коду програми як звичайного тексту та деяка обмеженість існуючих механізмів захисту авторського права, існує потреба у пошуку рішення, що дозволило б однозначно ідентифікувати порушення автентичності файлу вихідного коду із урахуванням особливостей предметної області.

На сьогоднішній день для проведення подібних перевірок найчастіше застосовуються алгоритми прямого порівняння, відстань Хеммінга та відстань Левенштейна. Основними недоліками цих методів для предметної області, що розглядається є обмежена кількість операцій, які можуть виявляти ці алгоритми та врахування регістру літер як різних символів. Як альтернативу даним методам було розглянуто використання алгоритму для визначення відстані Дамероу-Левенштейна. Даний алгоритм дозволяє виявляти застосування транспозиції при виконанні рефакторингу для файлу копії, тим самим показуючи значно кращі показники точності порівняння послідовностей. Однак, і даний метод має недоліки в контексті порівняння саме файлів вихідного коду, зокрема те ж врахування регістру літер як різних символів. В розрізі предметної області, що розглядається, така особливість може призводити до втрати точності порівняння, оскільки у коді регістр літер має вплив на зміст тексту значно менший, аніж у звичайному художньому творі, наприклад. Рішення для усунення даного недоліку було знайдено у зведенні усіх символів обох порівнюваних файлів вихідного коду до одного регістру перед початком перевірки.

Також, було виявлено необхідність пошуку рішення для оптимізації часу виконання порівняння, адже на реальних задачах із пошуку порушення автентичності файлів вихідного коду об’єми порівнюваних даних є досить великими, тому важливо

щоб алгоритм дозволяв проводити порівняння великого обсягу файлів за співставний чи менший час аніж уже використовувані механізми. Часова ефективність оригінального алгоритму для визначення відстані Дамероу-Левенштейна за Big O була визначена як $O(m * n)$, що свідчить про те, що час обробки вхідних даних алгоритмом повністю залежить від обсягу цих даних. В той же час було виявлено характерну особливість коду як текстової послідовності – він завжди матиме повторювані та невід'ємні елементи, що заздалегідь не впливають на результати порівняння, оскільки їх зміна призводить до втрати кодом працездатності. Таким чином, було запропоновано рішення додати перед початком обчислення відстані, обробку файлів перевірки алгоритмом, що видалятиме із послідовностей усі завчасно однакові елементи. Для перевірки працездатності запропонованих рішень було виконано обрахунки для однакової пари послідовностей за трьома алгоритмами: оригінальним алгоритмом для відстані Левенштейна, оригінальним алгоритмом для відстані Дамероу-Левенштейна та алгоритму для відстані Дамероу-Левенштейна із додаванням запропонованих рішень.

Для виконання перевірки було розроблено підсистему порівняння файлів вихідного коду, що дозволяє проводити обрахунки за трьома різними алгоритмами порівняння. Проведені тестування на основі десяти пар файлів вихідного коду різної символьної довжини та ступеню рефакторингу для файлу копії показали, що розроблене рішення дозволяє зберегти точність порівняння співставну із результатами отриманими за оригінальним алгоритмом для визначення відстані Дамероу-Левенштейна. Часова ефективність розробленого рішення досягнула приросту у 30% у порівнянні із приростом методу Дамероу-Левенштейна відносно алгоритму для відстані Левенштейна. Таким чином, похибка точності обчислення нівелюється. Основним недоліком даного рішення залишається те, що якщо до файлу копії було застосовано такий метод рефакторингу як зміна мови програмування яким написано код, то даний метод втрачає свою ефективність.

Отже, було підтверджено що розроблене рішення є ефективним покращенням існуючих методів порівняння для файлів вихідного коду програм за умови застосування до коду копії простих методів рефакторингу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бернська конвенція про охорону літературних і художніх творів (Паризький Акт від 24 липня 1971 року змінений 2 жовтня 1979 року) Rada Zakon URL: https://zakon.rada.gov.ua/cgi-bin/laws/main.cgi?nreg=995_051&print=1 (дата звернення: 22.05.2021).

2. Закон України «Про авторське право та суміжні права». Відомості Верховної Ради України (ВВР). 1994. № 13. ст.64

3. Інтелектуальна влісність в IT *Unionexpert* URL: http://unionexpert.su/intellektualnaya-sobstvennost-v-it/ (дата звернення: 22.05.2021).

4. Intellectual property rights when outsourcing software development *CShark* URL: https://cshark.com/blog/intellectual-property-rights-when-outsourcing-software-development/ (дата звернення: 22.05.2021).

5. Авторскі права: нотатки для програміста, ч. 1 *Dou* URL: https://dou.ua/lenta/articles/copyright-for-programmer/ (дата звернення: 22.05.2021).

6. Авторскі права: нотатки для програміста, ч. 4 *Dou* URL: https://dou.ua/lenta/articles/copyright-for-programmer-p4/ (дата звернення: 22.05.2021).

7. How to Avoid Intellectual Property (IP) Rights Traps in Software Development *Moqod* URL: https://moqod.com/how-to-avoid-intellectual-property-rights-traps-in-software-development/ (дата звернення: 22.05.2021).

8. Усе, що ви хотіли знати про авторське право в IT *Dou* URL: https://dou.ua/lenta/articles/copyright-in-it/ (дата звернення: 22.05.2021).

9. Version control *Wikipedia* URL: https://en.wikipedia.org/wiki/Version_control (дата звернення: 22.05.2021).

10. Using version control to manage Intellectual Property *Jisc* URL: https://osswatch.jiscinvolve.org/wp/2010/04/23/using-version-control-to-manage-intellectual-property/ (дата звернення: 22.05.2021).

11. Вступ в системи контролю версій *Hexlet* URL: https://ru.hexlet.io/courses/git_base/lessons/vcs_intro/theory_unit (дата звернення: 22.05.2021).

12. Subversion *Wikipedia* URL: https://en.wikipedia.org/wiki/Apache_Subversion (дата звернення: 22.05.2021).

13. Subversion (svn) - что это такое? *Hosting.com* URL: https://www.komtet.ru/lib/etc/svn/subversion-chto-eto-takoe (дата звернення: 22.05.2021).

14. Mercurial *Wikipedia* URL: https://en.wikipedia.org/wiki/Mercurial (дата звернення: 22.05.2021).

15. Git vs. Mercurial: How Are They Different? *Perforce* URL: https://www.perforce.com/blog/vcs/git-vs-mercurial-how-are-they-different (дата звернення: 22.05.2021).

16. The Myers diff algorithm: part 1 URL: https://blog.jcoglan.com/2017/02/12/the-myers-diff-algorithm-part-1/ (дата звернення: 22.05.2021).

17. Git *Wikipedia* URL: https://en.wikipedia.org/wiki/Git (дата звернення: 22.05.2021).

18. Що таке Git? *Atlassian* URL: https://www.atlassian.com/ru/git/tutorials/what-is-git (дата звернення: 22.05.2021).

19. Refactoring: Goals, Benefits and Why it’s Impartant *Langate* URL: https://dzone.com/articles/code-refactoring-techniques (дата звернення: 22.05.2021).

20. Code refactoring *Wikipedia* URL: https://en.wikipedia.org/wiki/Code_refactoring (дата звернення: 22.05.2021).

21. Прийоми рефакторингу Refactoring *Guru* URL: https://refactoring.guru/uk/refactoring/techniques (дата звернення: 22.05.2021).

22. Martin Fowler. Refactoring: Improving the Design of Existing Boston, 2019. 432 c.

23. String similarity — the basic know your algorithms guide! *ITNext* URL: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227 (дата звернення: 22.05.2021).

24. Про підходи до порівняння файлів PVSM : веб-сайт URL: https://www.pvsm.ru/razrabotka/6099#begin (дата звернення: 22.05.2021).

25. Непарний пошук в тексті і словнику *Habr* URL: https://habr.com/ru/post/114997/ (дата звернення: 22.05.2021).

26. Measuring Text Similarity Using the Levenshtein Distance *PaperspaceBlog* URL: https://blog.paperspace.com/measuring-text-similarity-using-levenshtein-distance/ (дата звернення: 22.05.2021).

27. Understanding the Levenshtein Distance Equation for Beginners *Medium* URL: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 (дата звернення: 22.05.2021).

28. Levenshtein Distance, in Three Flavors *People* URL: http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm (дата звернення: 22.05.2021).

29. The Levenshtein Distance Algorithm *DZone* URL: https://dzone.com/articles/the-levenshtein-algorithm-1 (дата звернення: 22.05.2021).

30. Levenshtein distance *Wikipedia* URL: https://en.wikipedia.org/wiki/Levenshtein_distance (дата звернення: 20.12.2021).

31. Часова складність *Wikipedia* URL: https://uk.wikipedia.org/wiki/%D0%A7%D0%B0%D1%81%D0%BE%D0%B2%D0%B0_%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D1%96%D1%81%D1%82%D1%8C (дата звернення: 22.05.2021).

32. Big O: Складність алгоритмів *The Code* URL: https://www.the-code.com.ua/skladnist-alghoritmiv/ (дата звернення: 22.05.2021).

33. Damerau–Levenshtein distance *Wikipedia* URL: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance (дата звернення: 22.05.2021).

34. Damerau-Levenshtein Edit Distance Explained *Lemida* URL: https://www.lemoda.net/text-fuzzy/damerau-levenshtein/ (дата звернення: 22.05.2021).

35. Damerau Levenshtein distance *Opengenus* URL: https://iq.opengenus.org/damerau-levenshtein-distance/ (дата звернення: 22.05.2021).

36. Мова C# і платформа .NET Core *Metanit* URL: https://metanit.com/sharp/tutorial/1.1.php (дата звернення: 22.05.2021).

37. Початок работи с .NET Framework *Microsoft.Docs* URL: https://docs.microsoft.com/ru-ru/dotnet/framework/get-started/ (дата звернення: 22.05.2021).

38. Windows Forms overview *Microsoft.Docs* URL: https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8 (дата звернення: 22.05.2021).

39. Кращі IDE для разробки на C# *GeekBrains* URL: https://gb.ru/posts/c_sharp_ides (дата звернення: 22.05.2021).

ДОДАТКИ

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління
інформаційною
безпекою" кафедри МБІС
д.т.н., професор
Юрій ЯРЕМЧУК
"15" вересня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:
Удосконалення методу виявлення порушення конфіденційності файлів вихідного коду за рахунок збільшення інтервалів обробки та застосування транспозиції
08-72.МКР.010.00.000.ТЗ

Керівник магістерської кваліфікаційної
роботи
к.т.н., доц., зав. каф. МБІС
Карпінець В. В.

Вінниця – 2022 р.

1. Найменування та область застосування

Програмний засіб реалізації методу порівняння файлів вихідного коду програм на основі алгоритму порівняння версій. Область застосування: захист файлів вихідного коду програм від несанкціонованого копіювання у системах зберігання файлів.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №203 від 14.09.2022 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективного методу порівняння файлів вихідного коду програм.

3.2 Призначення: розроблений програмний засіб має забезпечувати високу точність порівняння файлів вихідного коду та обчислення відповідної метрики.

4. Джерела розробки

4.1. Measuring Text Similarity Using the Levenshtein Distance *PaperspaceBlog* URL: https://blog.paperspace.com/measuring-text-similarity-using-levenshtein-distance/ (дата звернення: 20.12.2022).

4.2. Damerau Levenshtein distance *Opengenus* URL: https://iq.opengenus.org/damerau-levenshtein-distance/ (дата звернення: 20.12.2022).

4.3. Damerau-Levenshtein Edit Distance Explained *Lemida* URL: https://www.lemoda.net/text-fuzzy/damerau-levenshtein/ (дата звернення: 20.12.2022).

4.4. Big O: Складність алгоритмів *The Code* URL: https://www.the-code.com.ua/skladnist-alghoritmiv/ (дата звернення: 20.12.2022).

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.2

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1	Визначення напрямку магістерської роботи, формулювання теми	15.09.2022	30.09.2022
2	Аналіз предметної області обраної теми	01.10.2022	10.10.2022
3	Апробація отриманих результатів	11.10.2022	15.10.2022
4	Розробка алгоритму роботи	16.10.2022	31.10.2022
5	Написання магістерської роботи на основі розробленої теми	01.11.2022	15.11.2022
6	Розробка економічної частини	15.11.2022	23.11.2022
7	Передзахист магістерської кваліфікаційної роботи	24.11.2022	25.11.2022
8	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	26.11.2022	18.12.2022
9	Захист магістерської кваліфікаційної роботи	19.12.2022	21.12.2022

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;

– програмний додаток;

– презентація;

– відгук керівника роботи;

– відгук опонента

Технічне завдання до виконання прийняв _____ Ярова М. С.

Додаток Б – Блок-схеми алгоритмів

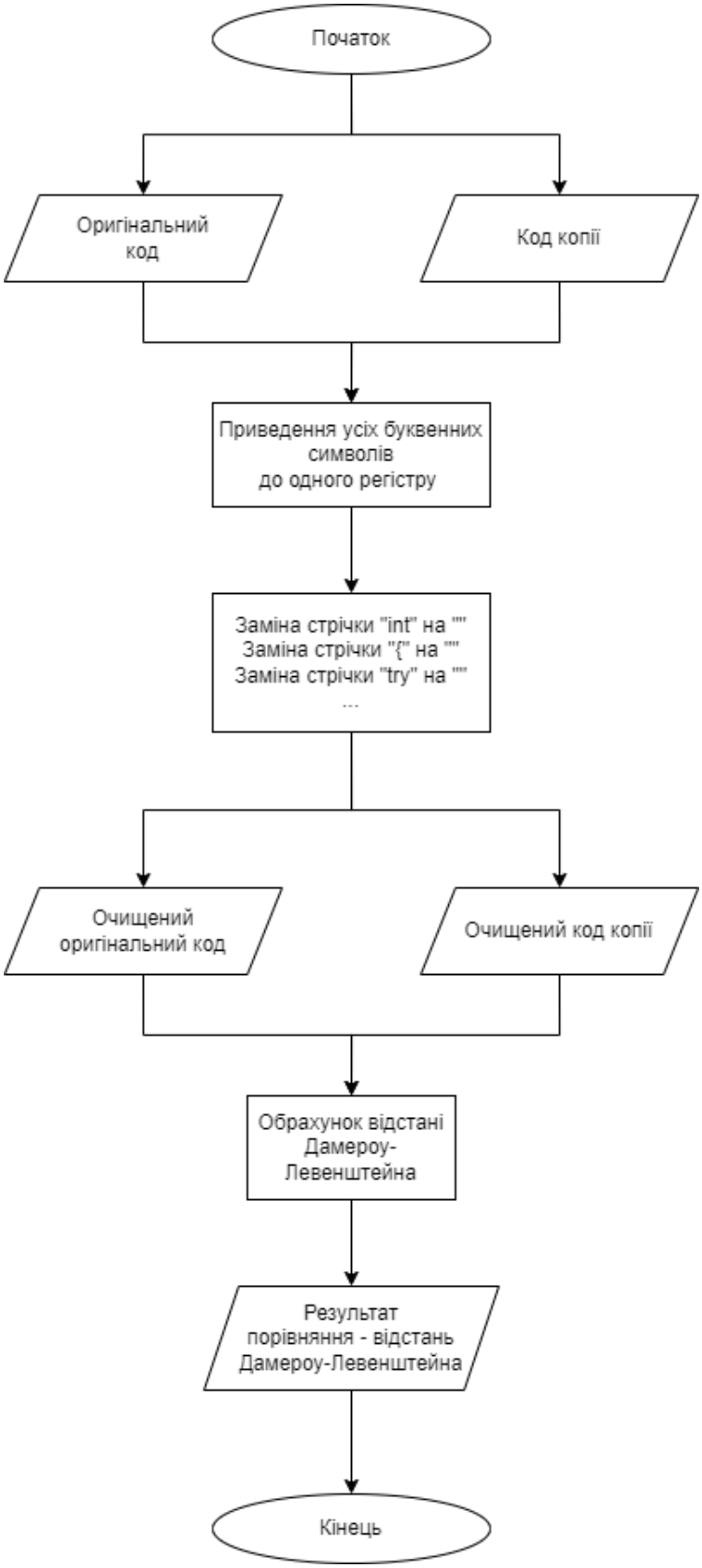


Рисунок Б1 - Структурна схема розробленого удосконалення до алгоритму пошуку відстані Дамероу-Левенштейна

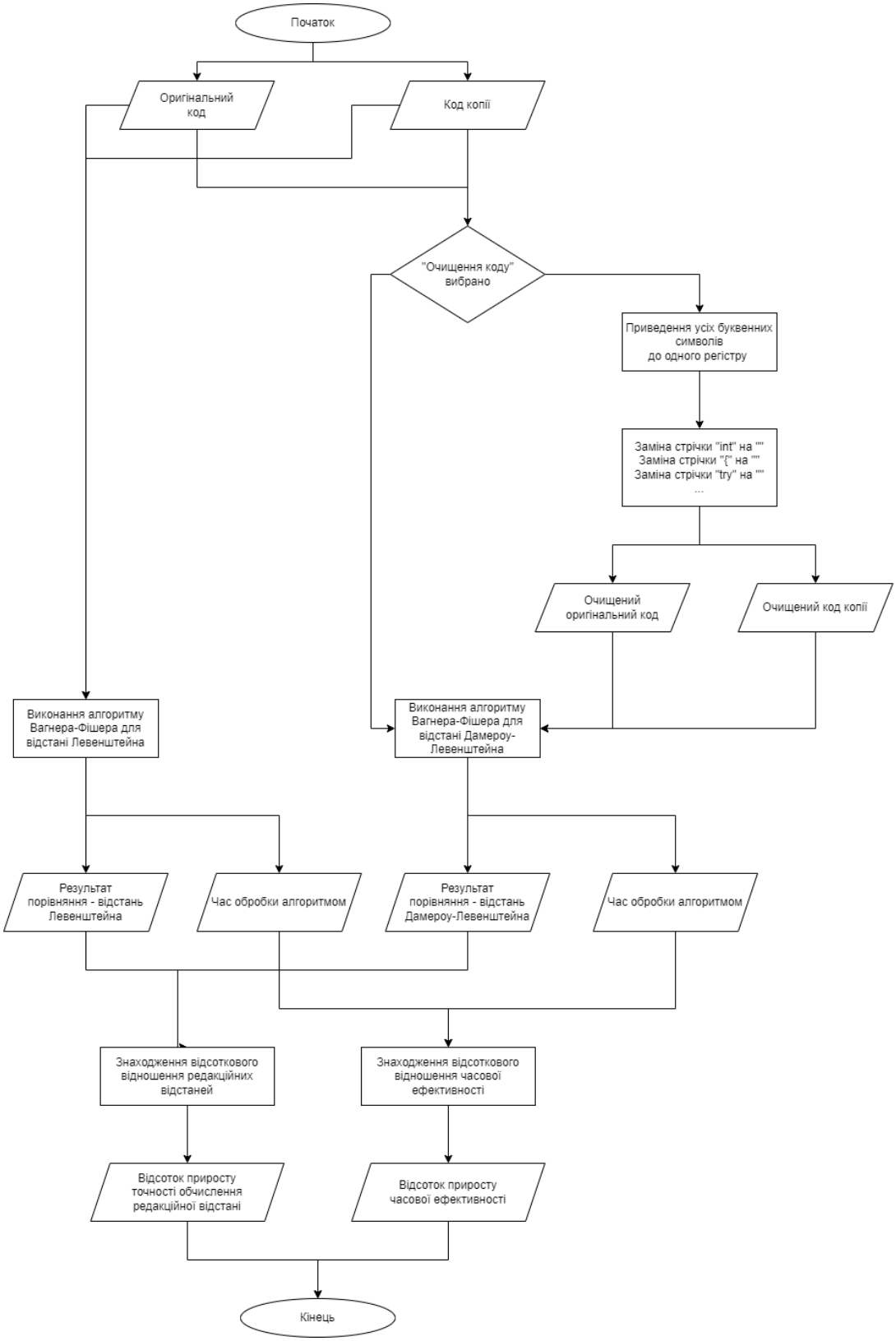


Рисунок Б2 - Блок схема алгоритму роботи реалізації підсистеми для тестування розробленого рішення

Додаток В – Лістинг коду розробленої підсистеми

```
namespace Diplom
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      long timer1Start = 0;
      long timer2Start = 0;
      long timer1End = 0;
      long timer2End = 0;
      private void button1_Click(object sender, EventArgs e)
      {
         richTextBox1.Text = "";
         if (openFileDialog1.ShowDialog() == DialogResult.Cancel) return;
         string filename = openFileDialog1.FileName;

         string[] lines = System.IO.File.ReadAllLines(filename);

         foreach (string line in lines)
         {
            richTextBox1.Text += line + "\n";
         }
      }

      private void button4_Click(object sender, EventArgs e)
      {
         richTextBox2.Text = "";
         if (openFileDialog1.ShowDialog() == DialogResult.Cancel) return;
         string filename = openFileDialog1.FileName;

         string[] lines = System.IO.File.ReadAllLines(filename);

         foreach (string line in lines)
         {
            richTextBox2.Text += line + "\n";
         }
      }

      private void button2_Click(object sender, EventArgs e)
      {
         levenstain();
      }

      void levenstain()
```

```
{
   string a = richTextBox1.Text;
   string b = richTextBox2.Text;

   Int32 dist = 0;

   timer1Start = TimeUtils.GetMicroseconds();

   if (string.IsNullOrEmpty(a))
   {
      if (!string.IsNullOrEmpty(b))
      {
         dist = b.Length;
      }
      dist = 0;
   }

   if (string.IsNullOrEmpty(b))
   {
      if (!string.IsNullOrEmpty(a))
      {
         dist = a.Length;
      }
      dist = 0;
   }
   Int32 cost;
   Int32[,] d = new Int32[a.Length + 1, b.Length + 1];
   Int32 min1;
   Int32 min2;
   Int32 min3;
   for (Int32 i = 0; i <= d.GetUpperBound(0); i += 1)
   {
      d[i, 0] = i;
   }
   for (Int32 i = 0; i <= d.GetUpperBound(1); i += 1)
   {
      d[0, i] = i;
   }
   for (Int32 i = 1; i <= d.GetUpperBound(0); i += 1)
   {
      for (Int32 j = 1; j <= d.GetUpperBound(1); j += 1)
      {
         cost = Convert.ToInt32(!(a[i - 1] == b[j - 1]));

         min1 = d[i - 1, j] + 1;
         min2 = d[i, j - 1] + 1;
         min3 = d[i - 1, j - 1] + cost;
         d[i, j] = Math.Min(Math.Min(min1, min2), min3);
      }
   }
```

```
        dist = d[d.GetUpperBound(0), d.GetUpperBound(1)];

        timer1End = TimeUtils.GetMicroseconds();
        label5.Text = dist.ToString();
        label8.Text = (timer1End - timer1Start).ToString();
        MessageBox.Show("Відстань Левенштейна: " + dist + ". Часу витрачено (млсек): " + (timer1End
- timer1Start).ToString());
    }

    private void button3_Click(object sender, EventArgs e)
    {
        modifiedLevenstein();
    }

    void modifiedLevenstein()
    {
        string m = "";
        string n = "";
        int[] current;
        int[] previous;
        int[] transposition;


        int max = 1024;
        int dist = 0;

        if (richTextBox1.Text.Length > richTextBox2.Text.Length)
        {
            m = richTextBox1.Text.ToUpper();
            n = richTextBox2.Text.ToUpper();
        }
        else
        {
            m = richTextBox2.Text.ToUpper();
            n = richTextBox1.Text.ToUpper();
        }

        timer2Start = TimeUtils.GetMicroseconds();

        if (checkBox1.Checked)
        {
            m = proceed(m);
            n = proceed(n);
        }
        int m_length = m.Length;
        int n_length = n.Length;

        int max_length;
        if (m_length > n_length) max_length = m_length;
        else max_length = n_length;
```

```
max = max_length;

current = new int[max_length + 1];
previous = new int[max_length + 1];
transposition = new int[max_length + 1];

if (m_length == 0)
   dist = n_length;

if (n_length == 0)
   dist = m_length;

if (m_length > n_length)
{
   string tmp = m;
   m = n;
   n = tmp;
   m_length = n_length;
   n_length = n.Length;
}

if (max < 0) max = n_length;
if (n_length - m_length > max)
   dist = max + 1;

if (m_length > current.Length)
{
   current = new int[m_length + 1];
   previous = new int[m_length + 1];
   transposition = new int[m_length + 1];
}

for (int i = 0; i <= m_length; i++)
   previous[i] = i;

char last_n_char = (char)0;
for (int i = 1; i <= n_length; i++)
{
   char n_char = n[i - 1];
   current[0] = i;

   // Обчислювати лише діагональну смугу шириною 2 * (max + 1)
   int from = Math.Max(i - max - 1, 1);
   int to = Math.Min(i + max + 1, m_length);

   char last_m_char = (char)0;
   for (int j = from; j <= to; j++)
   {
      char m_char = m[j - 1];
```

```
            int cost = m_char == n_char ? 0 : 1;
            int value = Math.Min(Math.Min(current[j - 1] + 1, previous[j] + 1), previous[j - 1] + cost);

            if (m_char == last_n_char && n_char == last_m_char)
               value = Math.Min(value, transposition[j - 2] + cost);

            current[j] = value;
            last_m_char = m_char;
         }
         last_n_char = n_char;

         int[] temporary = transposition;
         transposition = previous;
         previous = current;
         current = temporary;
      }

      dist = previous[m_length];

      timer2End = TimeUtils.GetMicroseconds();

      MessageBox.Show("Відстань Дамероу-Левенштейна: " + dist + ". Часу витрачено (млсек): " +
(timer2End - timer2Start).ToString());
      label6.Text = dist.ToString();
      label7.Text = (timer2End - timer2Start).ToString();
   }

   private string proceed(string inData)
   {
      inData = inData.Replace("INT32", "");
      inData = inData.Replace("INT", "");
      inData = inData.Replace(" ", "");
      inData = inData.Replace("\n", "");
      inData = inData.Replace("SYSTEM", "");
      inData = inData.Replace("STRING", "");
      inData = inData.Replace("VAR", "");
      inData = inData.Replace("WINDOWS", "");
      inData = inData.Replace("DOUBLE", "");
      inData = inData.Replace(".", "");
      inData = inData.Replace("(", "");
      inData = inData.Replace(")", "");
      return inData;
   }

   private void button5_Click(object sender, EventArgs e)
   {
      count();
   }
```

```
    void count()
    {
        int origOps = Convert.ToInt32(label8.Text);
        int modOps = Convert.ToInt32(label7.Text);
        float varOps = 100 - (float)100 * ((float)modOps / (float)origOps);
        label11.Text = varOps.ToString();

        int origDst = Convert.ToInt32(label5.Text
        int modDst = Convert.ToInt32(label6.Text);
        float varDst = 100 - (float)100 * ((float)modDst / (float)origDst);
        label12.Text = varDst.ToString();
    }
    public static class TimeUtils
    {
        public static long GetMicroseconds()
        {
            double timestamp = Stopwatch.GetTimestamp();
            double microseconds = 1_000_000.0 * timestamp / Stopwatch.Frequency;

            return (long)microseconds;
        }
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        label5.Text = "";
        label6.Text = "";
        label7.Text = "";
        label8.Text = "";
        label11.Text = "";
        label12.Text = "";
    }

  }
}
```

Додаток Г - Ілюстративний матеріал (презентація)

Презентація до захисту магістерської дипломної роботи на тему:

УДОСКОНАЛЕННЯ МЕТОДУ ВИЯВЛЕННЯ ПОРУШЕННЯ КОНФІДЕНЦІЙНОСТІ ФАЙЛІВ ВИХІДНОГО КОДУ ЗА РАХУНОК ЗБІЛЬШЕННЯ ІНТЕРВАЛІВ ОБРОБКИ ТА ЗАСТОСУВАННЯ ТРАНСПОЗИЦІЇ

Виконала студентка групи УБ-21м
Факультету менеджменту та інформаційної безпеки
Ярова Марія

ВСТУП

Метою дипломної роботи є удосконалення методу виявлення порушення конфіденційності файлів вихідного коду програм.

Об’єкт дослідження – процес захисту від несанкціонованого копіювання порушення автентичності файлів вихідного коду програм.

Предмет дослідження – методи та засоби порівняння версій файлів вихідного коду програм.

Новизна одержаних результатів. Застосовано метод порівняння версій файлів на основі алгоритму для визначення відстані Дамероу-Левенштейна у нетиповій предметній області та розроблено удосконалення до його реалізації для даної предметної області.

Практичне значення одержаних результатів. Розроблено програмну підсистему для порівняння версій файлів з метою доведення чи спростування факту порушення автентичності файлів вихідного коду програм та вчинення факту несанкціонованого копіювання, що також може бути інтегрована до більшої програмної системи захисту.

АКТУАЛЬНІСТЬ ТЕМИ

ПРОБЛЕМИ ЗАХИСТУ АВТОРСЬКОГО ПРАВА НА ВИХІДНИЙ КОД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- застосування для приховування факту запозичення технології рефакторингу коду.

- слабкі механізми ідентифікації порушення авторського права в існуючих системах управління версіями.

- відсутність надійних механізмів порівняння, що на основі чіткої метрики могли б оцінювати ступінь близькості порівнюваних послідовностей.

АЛГОРИТМИ ПОРІВНЯННЯ ВЕРСІЙ

- алгоритм прямого порівняння;
- відстань Хеммінга;
- відстань Левенштейна;
- відстань Дамероу-Левенштейна

АЛГОРИТМ ДАМЕРОУ-ЛЕВЕНШТЕЙНА

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), \text{якщо } \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{якщо } i,j > 1 \text{ і } a_i = b_{j-1} \text{ і } a_{i-1} = b_j \\ \text{інакше,} \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases}$$

$d_{a,b}(i-1,j)+1$ – видалення символу в послідовності a з перенесенням результату в b;

$d_{a,b}(i,j-1)+1$ – додавання символу в послідовність a з перенесенням результату в b;

$d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}$ – неспівпадіння символів у послідовностях a та b;

$d_{a,b}(i-2,j-2)+1$ – перестановка двох послідовних символів послідовності a з перенесенням результату в b

ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ УДОСКОНАЛЕННЯ

КРИТЕРІЇ ОЦІНКИ УСПІШНОСТІ УДОСКОНАЛЕННЯ АЛГОРИТМУ

- відношення результату обрахунку редакційної відстані за удосконаленою версією відстані Дамероу-Левенштейна до результатів отриманих при порівнянні тих же зразків коду за відстанню Левенштейна, як показник покращення точності визначення редакційної відстані.

- покращення показників швидкості обробки удосконаленим алгоритмом порівняння версій коду за відстанню Дамероу-Левенштейна у порівнянні зі швидкістю обробки порівняння за відстанню Левенштейна.

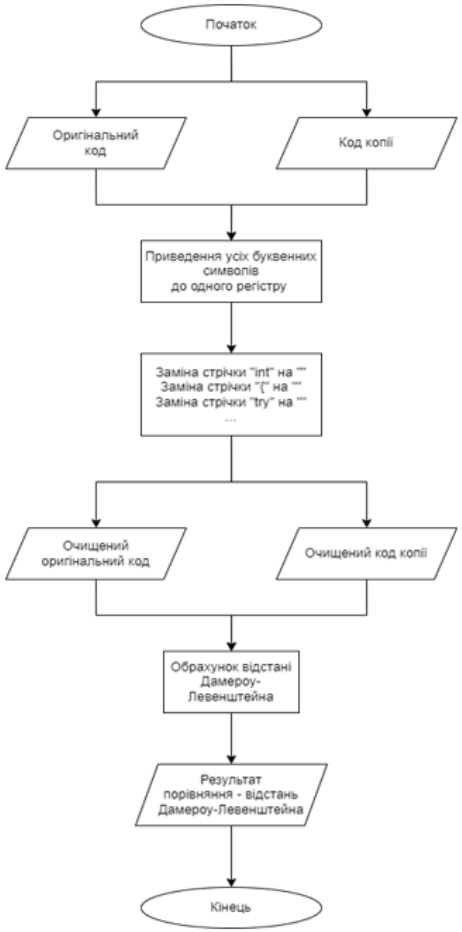
ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ УДОСКОНАЛЕННЯ

ЗАПРОПОНОВАНЕ РІШЕННЯ З УДОСКОНАЛЕННЯ

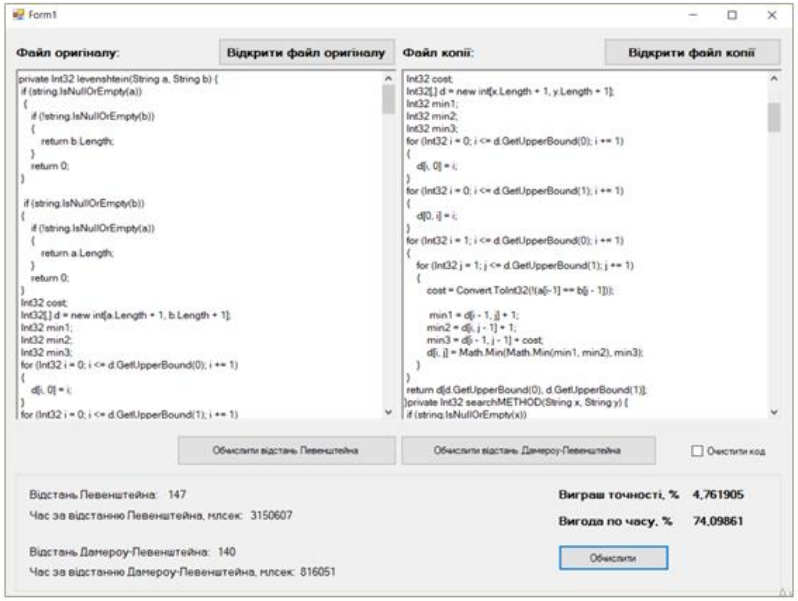
- Переведення символів до одного регістру.

- Видалення з послідовності усіх повторюваних, невід’ємних від коду символьних послідовностей та виразів, відповідно до мови програмування, на якій написано код.

СТРУКТУРНА СХЕМА РІШЕННЯ З УДОСКОНАЛЕННЯ



ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО РІШЕННЯ З УДОСКОНАЛЕННЯ



РЕЗУЛЬТАТИ ТЕСТУВАННЯ ЗАПРОПОНОВАНОГО РІШЕННЯ З УДОСКОНАЛЕННЯ

ТОЧНІСТЬ ПОРІВНЯННЯ

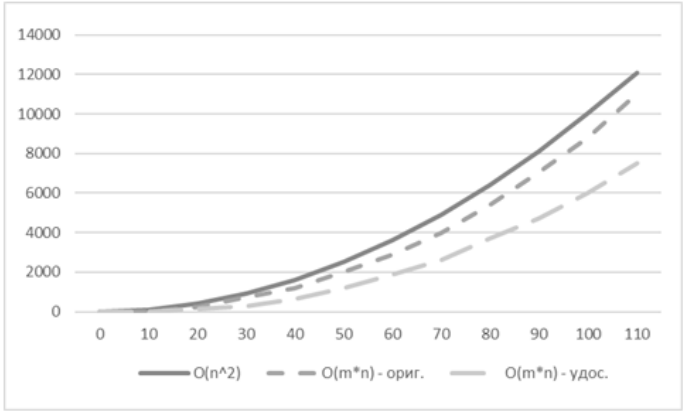
- У всіх тестах вдалося досягнути більшої точності порівняння послідовностей по відношенню до оригінального алгоритму для відстані Левенштейна.

- Не виявлено закономірностей щодо виграшу у точності порівняння. Це пояснюється тим, що на даний показник впливають одразу 2 фактори: довжина порівнюваних послідовностей а кількістю символів та степінь зміни файлу копії відносно оригіналу шляхом рефакторингу

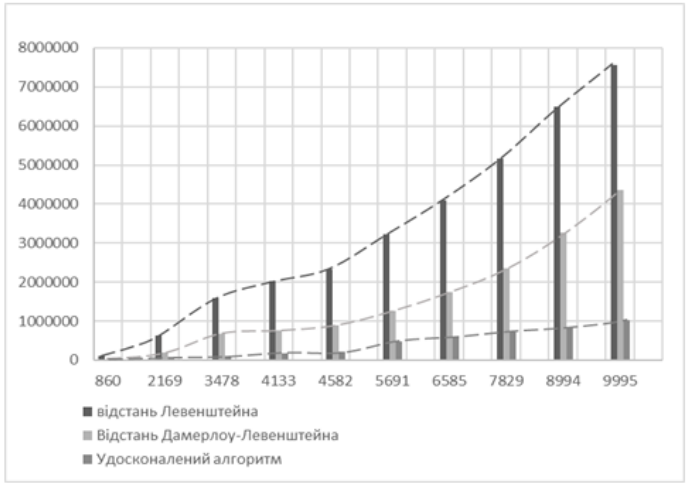
- Похибка обчислень редакційної відстані при порівнянні за удосконаленим алгоритмом $\Delta \approx 7{,}94\%$.

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ЗАПРОПОНОВАНОГО РІШЕННЯ З УДОСКОНАЛЕННЯ

ЧАСОВА ЕФЕКТИВНІСТЬ АЛГОРИТМУ



Очікуваний приріст



Приріст швидкості обробки за результатами тестування

ВИСНОВКИ

У даній роботі було виконано розробку рішення для удосконалення методу виявлення порушення автентичності у файлах вихідного коду програм.

Як альтернативу методам що вже використовуються для виконання перевірки подібності, було розглянуто використання алгоритму для визначення відстані Дамероу-Левенштейна. Даний алгоритм дозволяє виявляти застосування транспозиції при виконанні рефакторингу для файлу копії, тим самим показуючи значно кращі показники точності порівняння послідовностей.

Однак, і даний метод має недоліки в контексті порівняння саме файлів вихідного коду, зокрема те ж врахування регістру літер як різних символів. В розрізі предметної області, що розглядається, така особливість може призводити до втрати точності порівняння, оскільки у коді регістр літер має вплив на зміст тексту значно менший, аніж у звичайному художньому творі, наприклад. Рішення для усунення даного недоліку було знайдено у зведенні усіх символів обох порівнюваних файлів вихідного коду до одного регістру перед початком перевірки.

Також, було виявлено необхідність пошуку рішення для оптимізації часу виконання порівняння. було запропоновано рішення додати перед початком обчислення відстані, обробку файлів перевірки алгоритмом, що видалятиме із послідовностей усі завчасно однакові елементи.

ВИСНОВКИ

Для виконання перевірки було розроблено підсистему порівняння файлів вихідного коду, що дозволяє проводити обрахунки за трьома різними алгоритмами порівняння.

Проведені тестування на основі десяти пар файлів вихідного коду різної символьної довжини та ступеню рефакторингу для файлу копії показали, що розроблене рішення дозволяє зберегти точність порівняння співставну із результатами отриманими за оригінальним алгоритмом для визначення відстані Дамероу-Левенштейна. Часова ефективність розробленого рішення досягнула приросту у 30% у порівнянні із приростом методу Дамероу-Левенштейна відносно алгоритму для відстані Левенштейна. Таким чином, похибка точності обчислення нівелюється. Основним недоліком даного рішення залишається те, що якщо до файлу копії було застосовано такий метод рефакторингу як зміна мови програмування яким написано код, то даний метод втрачає свою ефективність.

Отже, було підтверджено що розроблене рішення є ефективним покращенням існуючих методів порівняння для файлів вихідного коду програм за умови застосування до коду копії простих методів рефакторингу.

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Вдосконалення методу виявлення порушення конфіденційності файлів вихідного коду за рахунок збільшення інтервалів обробки та застосування транспозиції

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98 % Схожість 2 %

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Коваль Н.П. (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Ярова М.С. (прізвище, ініціали)

Керівник роботи _____ (підпис) Карпінець В.В. (прізвище, ініціали)